IMPLEMENTACION DE UN MOTOR DE RENDERIZADO CON UN ALGORITMO DE RAYTRACING UTILIZANDO ACELERACION MEDIANTE LOS RECURSOS DE GPU.

Autores:

JULIÁN MONTES FRANCO

LUIS FERNANDO MONCADA IDÁRRAGA

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACION

PEREIRA

2015

IMPLEMENTACION DE UN MOTOR DE RENDERIZADO CON UN ALGORITMO DE RAYTRACING UTILIZANDO ACELERACION MEDIANTE LOS RECURSOS DE GPU.

Autores:

JULIÁN MONTES FRANCO

LUIS FERNANDO MONCADA IDÁRRAGA

PROYECTO DE GRADO PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS Y COMPUTACIÓN

Director:

RAMIRO ANDRÉS BARRIOS VALENCIA

Ingeniero de Sistemas y Computación

Especialista en Electrónica Digital

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACION

PEREIRA

2015

Notas de Aceptación:

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Pereira 11 de Mayo de 2015

AGRADECIMIENTOS

Agradecemos a la Universidad Tecnológica de Pereira a nuestros familiares, amigos y compañeros que nos acompañaron durante todo el proceso y de igual manera agradecemos a todos los docentes que han compartido sus conocimientos haciendo posible nuestra formación y en especial a nuestro director Ramiro Andrés Barrios por su interés, confianza y colaboración.

CONTENIDO

Pág.

LISTA DE TABLAS

LISTA DE GRAFICOS

RESUMEN

En la actualidad las unidades de procesamiento grafico (GPU) disponen de una interfaz de programación, permitiendo ser utilizadas para ejecutar tareas de propósito general (GPGPU).

El trabajo se basó en comparar los resultados obtenidos de la renderización de imágenes compuestas por objetos sólidos, la primera parte fue un desarrollo sobre CPU y la segunda sobre GPU. El incremento que se ha dado en las aplicaciones multiproceso ha permitido que las GPUs evolucionen admitiendo en ellas el cálculo de operaciones que pueden o no estar relacionadas con temas gráficos.

Para el trabajo han sido utilizadas las GPU de la compañía NVIDIA y su entorno y lenguaje de desarrollo CUDA con el IDE Nsight de eclipse diseñado especialmente para este fin.

El desarrollo del proyecto se dividió en dos fases, la primera de estas fue el desarrollo sobre CPU el cual arrojo resultados satisfactorios permitiendo inclusive el manejo de texturas, las cuales en la segunda fase que consiste en el desarrollo sobre GPU se dejaron a un lado ya que CUDA no permite el manejo de texturas.

1. INTRODUCCIÓN

1.1. Presentación

Con el creciente desarrollo de la tecnología se volvió normal encontrar aplicaciones que requieren de equipos de cómputo de alto desempeño. Estas aplicaciones tienen principalmente relación con ciencias y tecnologías como: virtualización, modelado y simulación, aplicaciones de bases de datos, Bioinformática, por mencionar algunas. En busca de satisfacer esa necesidad cada día se encuentran nuevas tecnologías o dispositivos independientes que se encargan de una función específica, son conocidos como sistemas embebidos, los cuales ya se comunican e interactúan con un sistema multipropósitos como es el computador.

Con el tiempo se fue demostrando que ya no es suficiente tener altas frecuencias en un procesador, si no la capacidad de realizar varias tareas simultáneamente, por lo cual se empezaron a crear procesadores con dos núcleos, posteriormente cuatro y en la actualidad ese es el estándar para un computador hogareño, pero también es posible encontrar procesadores de más de 8 núcleos el problema está en que no todo el software viene diseñado para utilizar esos recursos y además la forma de acceder a estos es diferente según el fabricante.

En las unidades de procesamiento grafico o GPU por sus siglas en ingles se pueden encontrar grandes cantidades de microprocesadores que en la actualidad pueden ser utilizados no solo para cuestiones graficas si no que han ampliado su capacidad y permiten que el desarrollador acceda estos recursos, su fuerte son los gráficos debido a las grandes cantidades de datos que tienen que operar y para eso fueron creadas. Cuentan con una característica muy importante y es que los costos por microprocesador son económicos en comparación con los de una CPU; haciendo posible y viable que tanto en el campo científico como industrial se empiecen a usar estas tecnologías.

El documento se encuentra divido por cinco capítulos, de los cuales en el primero de estos encontrará toda la justificación y presentación del proyecto.
En el segundo capítulo se encuentra toda la información y los conceptos necesarios para la comprensión de todo el documento, los antecedentes y las características principales de los algoritmos utilizados para hacer renderizado.
En el tercero se hace el análisis comparativo, se detallan los resultados obtenidos y las diferencias entre ambos tipos de ejecuciones incluyendo la información de cómo fue desarrollado el proyecto, para que finalmente en el cuarto capítulo se incluyan las conclusiones, las recomendaciones y el trabajo futuro que surge a partir del presente proyecto.
El quinto y último capitulo contiene las referencias bibliográficas de la información consultada para la realización del proyecto.

1.2. Planteamiento del problema

Con la evolución de la computación, las aplicaciones multimedia se fueron convirtiendo en algo importante en los sistemas informáticos, buscando facilidad en el manejo del software, conectividad con otros dispositivos y no solamente el procesamiento de los datos obtenidos sino también su presentación.

Un video es el procesamiento, almacenamiento y transmisión de imágenes, esto se daba por medios analógicos y en la actualidad por medios digitales, representando el movimiento con un conjunto de imágenes, las cuales surgen bajo dos opciones, una de estas es una fotografía, lo cual consiste en captar la luz por medio de un sensor utilizando una cámara y la segunda opción es un procesamiento digital a partir de un modelo de datos, esto se convierte en la materia prima para industrias como el cine.

En la actualidad se pueden encontrar varios algoritmos que realizan esta función pero el ***Raytracing*** presenta buenos resultados a un costo computacional[3] elevado, en donde no es aprovechado la capacidad total de un equipo, esto se da por la utilización de efectos como la reflexión y la refracción, efectos que muchos otros algoritmos no implementan o si lo hacen surge de simulaciones generando resultados poco precisos.

Tanto las imágenes como el mundo a nuestro alrededor están compuestos por diferentes objetos; sólidos, líquidos, gaseosos y fuentes de luz (esta última puede ser de varios tipos). El algoritmo consiste en modelar el comportamiento de la luz y su interacción con el mundo; en el caso particular del proyecto se modelarán los objetos sólidos.

El proyecto será realizado utilizando la arquitectura de cálculo en paralelo CUDA[1] (Arquitectura Unificada de Dispositivos de Computo) la cual es un desarrollo de la empresa NVIDIA[2] pionera en el campo de las GPU (Unidades de Procesamiento Gráfico).

[0] http://www.portafolio.co/negocios/negocio-del-cine-colombia

[1] *Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo)*

[2] http://la.nvidia.com/page/home.html

1.3. Justificación

Con el proyecto se pretende resaltar las ventajas que presenta la ejecución de este tipo de algoritmos de renderizado de imágenes sobre GPU´s, ya que en estos dispositivos se encuentra recursos computacionales a bajos costos, permitiendo que tanto las compañías de los sectores involucrados como las que necesiten realizar este tipo de procesos computacionales puedan acceder a este servicio de una forma rápida y económica.

La evolución de los computadores ha traído con sigo una mayor cantidad de recursos inclusive en las máquinas hogareñas, permitiendo que en la actualidad haya una gran afluencia multimedia, donde dejo ser un tema específico de los diseñadores, arquitectos, cineastas; considerándose fundamental para los procesos educativos, un ejemplo sencillo es el actual sistema operativo de la empresa Microsoft – Windows 8, y su antecesor Windows 7 [4] , donde su cambio más radical ha sido la optimización de éste hacia lo multimedia, ampliando formatos compatibles, mejorando las imágenes, entre otros (resaltando de igual manera los todos los cambios internos y mejoras que incluye).

El proceso de generar una imagen o renderizado como es conocido lo utilizamos casi en todo momento que interactuamos con un computador o un dispositivo móvil muchas veces sin percatarnos de esto; cualquier tipo de edición a una imagen que realizamos requiere practicar un renderizado, por tal motivo las aplicaciones usualmente reducen el tamaño del archivo para así disminuir la complejidad del proceso a realizar.

Se busca igualmente que con el proyecto se potencialice e incremente el uso de este tipo de tecnologías en Colombia, dado que en algunos países ya se ha venido explotando, tanto en campos científicos como en algunas empresas; de igual manera siempre se está en un continuo proceso de mejora donde se busca brindar un servicio ágil y eficiente en todo tipo de procesos con especialidad en los relacionados con la informática.

[3] http://www.cs.berkeley.edu/~luca/notes/complexitynotes02.pdf
[4] http://www.samsung.com/es/article/mejoras-de-windows-8-frente-a-windows-7/

1.4. Objetivos

1.4.1. Objetivo General:

Implementar un motor de renderizado con un algoritmo de Raytracing en donde se utilicen recursos de GPU.

1.4.2. Objetivos Específicos:

- ✓ Realizar un estudio de algoritmos de renderizado para la implementación sobre CPU.
- ✓ Probar diferentes características que pueden o no ser agregadas al algoritmo.
- ✓ Implementar el algoritmo sobre CPU.
- ✓ Paralelizar del algoritmo para la implementación sobre GPU.
- ✓ Comparar los resultados obtenidos en las dos implementaciones.

1.5. Alcance

Dentro de las ventajas que se pueden obtener al implementar una aceleración de un algoritmo de renderizado como el Raytracing se pueden enunciar las siguientes:

- ✓ Disminuir los tiempos de ejecución sin disminuir la calidad de los resultados.
- ✓ Servir de herramienta a proyectos que requieran de procesos de renderización de alta demanda computacional.
- ✓ Agilizar el desarrollo de proyectos relacionados mejorando los tiempos de procesamiento.
- ✓ Presentar una oportunidad para que personas externas a la universidad o empresas puedan acceder a la herramienta como un servicio.
- ✓ Disminuir costos por obtener tiempos de ejecuciones más cortos.

2. MARCO REFERENCIAL

2.1. Marco Conceptual

- Comportamiento de la luz: La luz como producto de la radiación electromagnética presenta comportamientos de onda (caracterizadas por tener longitud, frecuencia, amplitud) y su comportamiento puede ser reflexión, absorción, refracción.
- Luz indirecta: Es la emitida por los objetos que rodean al que se está estudiando, esta luz no considerada fuente de luz si no una reflexión de esta.
- Cámara y escena: Se hace referencia al observador o al punto desde donde se observa la pantalla, su función consiste en definir los tipos de proyección (perspectiva o paralela).
- Puntos de referencia visual: Es un punto variable donde se sitúa el puerto de vista, siendo inicialmente centrado se calcula a través de las coordenadas de los objetos.
- Ventana: Es un cuadro que delimita el plano de proyección, debido a que los espacios de presentación son limitados y el tamaño de la escena es fijo.
- Rayo de luz: Es la línea imaginaria que representa el desplazamiento de la luz, se lanzaran rayos de luz desde la cámara u observador para calcular la interacción de estos con la escena; es un vector situado en un punto.
- Punto: Es representado bajo dos coordenadas (x, y) en un plano 2D y bajo 3 coordenadas (x, y, z) en 3D.
- Escena: Es una descripción geométrica limitada a unos objetos (sólidos en este caso) y una cámara (también conocido como observador).
- Ejes: Representan las coordenadas para el desplazamiento en la imagen.

- Planos de proyección: Es el plano (X, Y) donde se plasma la escena con la característica de ser infinito.

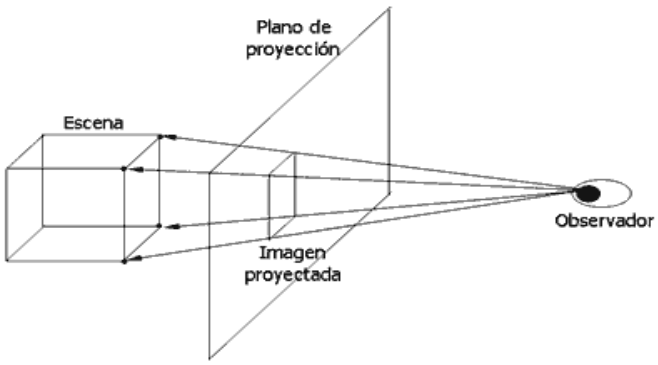


Ilustración 1: Plano de proyección

- Arquitectura de una GPU: Una GPU varía de una CPU en que la primera se compone de muchos microprocesadores, lo cual permite aceleraciones en algunos programas o algoritmos.

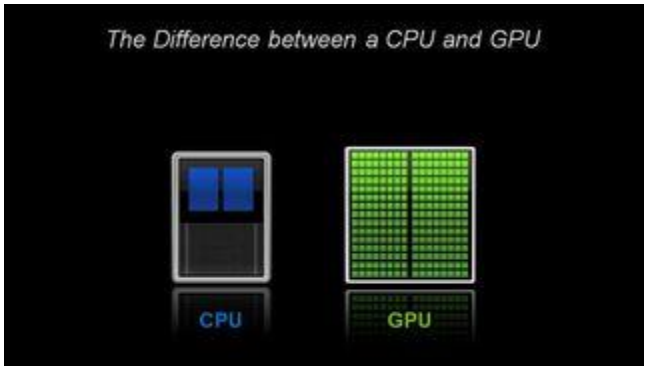


Ilustración 2: CPU vs GPU

- Memoria: En CUDA se manejan varios tipos de memoria la cual es completamente independiente a la del Host, el primero de estos es la memoria global, la cual es la más pobre en cuestiones de rendimiento presentando largas latencias en el acceso, la memoria constante solamente es usada para operaciones de lectura realizadas por el Host, presenta mejores rendimientos que la memoria global, los registros y la memoria compartida son accedidos de maneras rápidas, los registros son asignados a cada hilo y la memoria compartida es asignada a conjuntos de hilos; la última versión de CUDA incorpora una memoria unificada la cual permite al Host y al Device tener una abstracción de manera que la memoria entre ambos parece ser una misma, facilitando la programación.

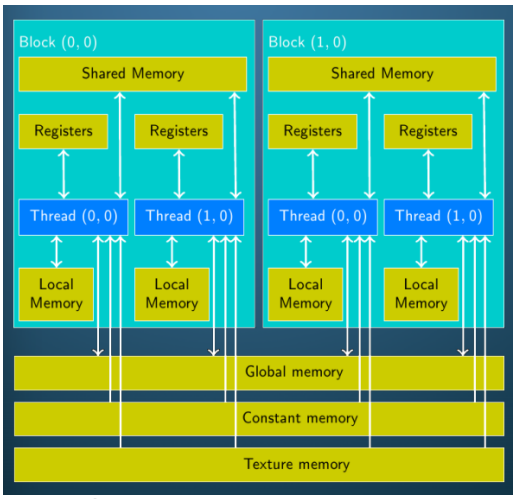


Ilustración 3: Memoria en una GPU

- Estructura de un programa en CUDA: Inicialmente un programa en CUDA es como cualquier software, su ejecución o sus primeras sentencias son ejecutadas en el Host (CPU) al igual que las etapas que tienen poco o nulo paralelismo, aquellas que presentan paralelismo se implementan para que su ejecución sea en el Device (GPU) el compilador C de Nvidia (nvcc) separa ambos tipos, compilando el código C convencional con el compilador del Host y el código para ejecutar en la GPU que está escrito en C con algunas extensiones es compilado por el nvcc el cual permiten etiquetar las funciones paralelas, que son llamadas Kernels. Los kernels generan un determinado número de hilos que se caracterizan por ser ligeros en comparación con los de la CPU necesitando pocos ciclos del reloj para ser generados y programados.

- Organización de los Hilos en CUDA: Los hilos son organizados en dos niveles usando sus coordenadas para la indexación, esto se da con las palabras "blockId" y "threadId", en donde la primera representa todos los hilos de un mismo bloque y su rango viene definido por el tamaño de la malla

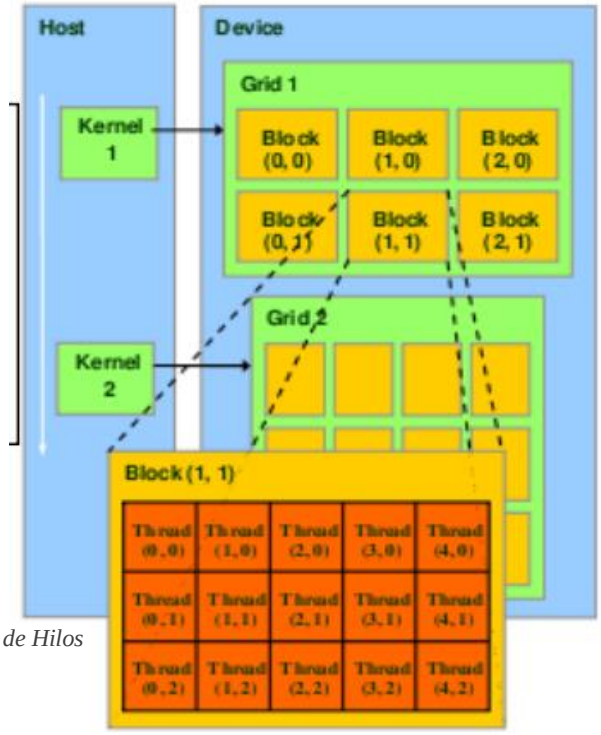


Ilustración 4: Modelo de Hilos en CUDA

Con este código se define una malla bidimensional de 2x2 bloques y bloques tridimensionales de 4x2x2 hilos, y la última línea es la encargada de iniciar el kernel.

```
dim3 dimBlock(4, 2, 2);
dim3 dimGrid(2, 2, 1);
KernelFunction<<<dimGrid, dimBlock>>>(...);
```

Ilustración 5: Ejemplo creación de hilos

- Programación en Paralelo: Este término hace referencia a que un programa puede realizar operaciones aritméticas sobre una estructura de datos de forma segura, simultánea y de manera independiente, un ejemplo puede ser la multiplicación de dos matrices, en donde se accede elemento por elemento de la fila de una matriz A y éste se multiplica por el mismo elemento en la columna de la matriz B, cada resultado se obtiene de manera independiente de los demás.

- Reflexión especular: Es el rebote de la luz al llegar a un objeto, conservando una alta intensidad de la luz, obteniendo como resultado una mancha más clara en la superficie del objeto.

- Renderizar: Hace referencia al proceso de generar una imagen o un video mediante el cálculo de la iluminación, partiendo de un modelo en 3D, siguiendo los rayos de luz por los que una cámara u observador es atravesado, analizando la incidencia de estos con los objetos.
- Rotación: Desplazamiento de un punto pero en ángulos y no en distancias.

$$\begin{bmatrix} \cos\beta & -\sin\beta & 0 & 0 \\ \sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotación respecto Z

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotación respecto X

$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotación respecto Y

Tabla 1: Matriz de rotación

- Reflexión: Es el cambio de dirección de una onda.
- Refracción: Es el cambio de dirección que sufre una onda al cambiar de un material a otro, en el caso de la luz se da por los cambios en la densidad óptica.

2.2. Antecedentes

Durante muchos años, se contó con procesadores mono núcleos, los cuales en su época realizaban las funciones necesarias, con el pasar del tiempo se fue requiriendo mayores cálculos computacionales, los cuales hacían que los procesadores aumentaran la frecuencia de funcionamiento o clock como es conocido, hasta que se concluyó que se obtendrían mejores resultados no solo aumentando la frecuencia de funcionamiento si no la cantidad núcleos, desde ese momento se empezó a hablar de multitarea y multiprocesador, lo cual consiste en realizar varias tareas simultáneamente.

Siempre se ha buscado mejorar el rendimiento de las aplicaciones, siendo este el fuerte del procesamiento paralelo, empresas como Adobe, Ansys, Wolfram Research están utilizando en mayor medida los recursos que brindan las GPUs

CUDA[5] llega al mercado en noviembre del año 2006, donde la compañía NVIDIA saco al mercado la Geforce 8800 GTX la cual soporta la computación de carácter general.

El término de RayTracing[6] o trazado de rayos aparece en los años de 1637, cuando Rene Descartes lo utiliza para tratar de comprender la formación del arco iris; posteriormente es Newton quien amplia la teoría para explicar igualmente los colores del arco iris.

En los años 1968 es presentado el primer algoritmo de raytracing, el cual posteriormente fue denominado raycasting debido al manejo de los reflejos.

2.3. Marco Teórico

2.3.1. Computación Grafica

Según el libro Computer Graphics de los autores John F. Hugues, Andries Van Dam, Morgan Mcguire, David Sklar, James Foley, Steven Feiner y Kurt Akeley. Definen la computación grafica como el campo de la informática en el que interviene todo lo visual, la ciencia y arte de la comunicación visual caracterizada por utilizar computadores y los dispositivos que interactúan con este.

En los principios de la computación se hablaba de una vía de comunicación que es la que se daba entre el hombre y el computador, esto ocurre a través de dispositivos como el ratón y el teclado, pero en la computación grafica cambio esta vía de comunicación y permite que a través de pantallas se muestre información de respuesta a quien usa el computador.

5 http://la.nvidia.com/object/cuda_home_new_la.html
6http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/modelosIlumionacion/raytracing_introduccion.html

La computación grafica es una disciplina compleja donde interviene campos como la física, la matemática, la percepción humana, la interacción hombre máquina, la ingeniería, el diseño gráfico y juega importantes roles en la vida de todas las personas; se usa la física para modelar la luz y mejorar la simulaciones de las animaciones, se usa la matemática para describir formas, la percepción humana para determinar lo que es relevante y/o pueda significar algo evitando así gastar tiempo renderizado algo que no vaya a ser necesario; se usa la ingeniería para optimizar todos los recursos que son necesarios (memoria, procesador , ancho de banda) y finalmente el diseño gráfico y las artes combinadas con la interacción hombre maquina son la llave fundamental para que exista la nueva vía de comunicación máquina-humano.

En el campo de la computación grafica se utiliza la palabra “modelo” la cual puede hacer referencia a el modelo geométrico o el modelo matemático, en donde el modelo geométrico es el modelo que como desarrollador del proyecto se desea que aparezca en la imagen, por ejemplo el modelo de una casa, de un avión, de un objeto y este modelo está compuesto por una gran variedad de atributos como el color, la textura la reflectancia; El modelo matemático es un modelo físico o computacional que describe como el objeto se mueve o como está afectado por las condiciones de luz, la cámara y el observador.

Gran parte de la investigación actual en los gráficos es en los métodos para la creación de modelos geométricos, métodos para la representación de reflectancia de la superficie (y reflectancia del subsuelo, y reflectancias de medios de participantes como la niebla y el humo, etc.), la animación de escenas por leyes y por aproximaciones de esas leyes físicas, el control de la animación, interacción con objetos virtuales, y, en los últimos años, una creciente integración de las técnicas de visión por computador. Como resultado, los campos de la computación gráfica y visión por computador están creciendo cada vez más cerca el uno al otro.

- **Historia**

Los primeros investigadores de la computación Grafica trabajaban en un contexto con recursos limitados, es decir procesadores con frecuencias bajas; los primeros esfuerzos estaban basados en realizar dibujos y tratando de hacer imágenes (por ejemplo imágenes foto realistas) en ese caso se hicieron muchos supuestos en consecuencia de los procesadores disponibles y las tecnologías de pantalla del momento, en momentos donde un simple monitor costaba más que el salario de un ingeniero, cada imagen mostrada tenía que tener algún valor.

Cuando las velocidades de los procesadores fueron medidas en MIPS (por sus siglas en ingles que significan millones de instrucciones por segundo) pero las imágenes contaban con 250.000 o 500.000 pixeles no se podía soportar rendimiento para todos los cálculos por pixel.

Gradualmente los modelos se fueron convirtiendo más ricos en formas, en luces y en reflexiones, pero incluso en la actualidad el modelo que describe la luz en la escena incluye el término ambiente, el cual significa cierta cantidad de luz que está en algún lugar de la escena sin ningún origen claro.

las pantallas han mejorado enormemente en los últimos años, con un cambio de dispositivos-vector a dispositivos-raster que muestran una serie de pequeños puntos, por ejemplo, al igual que los CRT o LCD pantallas en los años 1970 a 1980, y con el constante y lento aumento de la resolución (la pequeñez de los puntos individuales), tamaño (la dimensiones físicas de las pantallas), y rango dinámico (la relación de los valores más brillantes de los píxeles a los más débiles posibles); en los últimos 25 años. El desempeño de los procesadores gráficos también ha progresado de conformidad con la Ley de Moore (la tasa de mejora exponencial ha sido mayor para los procesadores gráficos que para CPUs). Las Arquitectura de procesamiento de gráficos también crece de manera paralela.

En ambos procesadores y pantallas, también han habido saltos importantes de progreso a lo largo del tiempo: El cambio de pantallas-vector a pantallas-raster y la rápida acogida del mercado de computadores portátiles y estación de trabajo, ha sido una de las razones

- **Características de los monitores y resolución del ojo**

Como es necesario el trabajo con monitores para ver la información del computador es importante conocer un poco respecto a esto; un monitor típico en el año 2010 tenía entre 1 millón y 1.5 millones de pixeles (partes individuales controlables del monitor) y con los años han crecido superando los 4 millones de pixeles en pantallas de no más de 15 pulgadas.

El asunto con estas resoluciones, así sea de una pantalla o una fotografía, es que pueden ser medidas con exactitud porque consisten en imágenes fijas. Por ejemplo, el monitor de 1600×1200 píxeles de resolución está dentro de un marco de 23 pulgadas, lo que hace que la resolución sea específica y completamente fija, eliminando la posibilidad de aumentar la resolución usando ese mismo monitor.

Por estas razones el ojo humano no puede tener una resolución exacta porque la visión no está enfrascada en una imagen fija. Al contrario, es posible mover los ojos hacia los lados, hacia arriba y hacia abajo, creando un campo de visión mucho más grande y con distintos grados de inclinación, y además, también es posible mover la cabeza en varias direcciones para capturar inclusive una mayor cantidad de imágenes.

Aunque no es posible tener una respuesta exacta de cuál es la resolución del ojo humano, utilizando algunos datos que ya se conocen sobre el ojo promedio, es posible promediar una resolución. Empezando por el hecho de que la retina tiene alrededor de cinco millones de conos receptores que son responsables de la visión a color. Con esta información se podría decir que los humanos tienen 5 megapíxeles de resolución, pero también hay que tomar en cuenta los 100 millones de bastones que detectan el contraste monocromático, que es sumamente importante para la nitidez de la imagen que se logra ver. Ahora se está hablando de 105 megapíxeles, un número que sigue subestimando a el ojo humano porque no se trata de una cámara que tome imágenes fijas.

No se puede dejar a un lado lo más importante, el movimiento de los ojos. Cada uno puede moverse horizontal un promedio de 120 grados, y verticalmente un promedio de 60 grados. Según Clarkvision[7] cada píxel es de alrededor 0.3 arcmin (minuto sexagesimal o de arco), y luego de algunas operaciones matemáticas donde se incluyen el movimiento promedio en grados de cada ojo, y los minutos de arco, es posible concluir que los ojos humanos tienen alrededor de 576 megapíxeles de resolución. La cámara más avanzada del planeta, la *Dark Energy*[8] *Camera* hecha por Fermilab, es de 570 megapíxeles, y, si algo es seguro es que no es tan compleja como el ojo humano.

- **GRÁFICOS EN 2D**

El primer avance en la computación gráfica fue la utilización del tubo de rayos catódicos. Hay dos acercamientos a la gráfica 2d: vector y gráficos. La gráfica de vector almacena datos geométricos precisos, topología y estilo como posiciones de coordenada de puntos, las uniones entre puntos (para formar líneas o trayectos) y el color, el grosor y posible relleno de las formas. La mayor parte de los sistemas de vectores gráficos también pueden usar primitivas geométricas de forma estándar como círculos y rectángulos etc. En la mayor parte de casos una imagen de vectores tiene que ser convertida a una imagen de trama o para ser vista. Los gráficos de tramas o (llamados comúnmente es una rejilla bidimensional uniforme de pixeles. Cada pixel tiene un valor específico como por ejemplo brillo, transparencia en color o una combinación de tales valores. Una imagen de trama tiene una resolución finita de un número específico de filas y columnas. Las demostraciones de computadora estándares muestran una imagen de trama de resoluciones como 1280 (columnas) x 1024 (filas) pixeles. Hoy uno a menudo combina la trama y lo gráficos vectorizados en formatos de archivo compuestos pdf, swf, svg

[7]http://www.clarkvision.com/articles/eye-resolution.html
[8]http://chicago.cbslocal.com/2013/09/04/fermilabs-dark-energy-camera-one-of-the-most-powerful-in-the-world/

- **GRÁFICOS EN 3D**

El término gráficos 3D por computadora hace referencia a trabajos de arte gráfico que son creados con ayuda de computadoras y software especial. En general, el término puede referirse también al proceso de crear dichos gráficos, o el campo de estudio de técnicas y tecnología relacionadas con los gráficos tridimensionales. Un gráfico 3D difiere de uno bidimensional principalmente por la forma en que ha sido generado.

Este tipo de gráficos se originan mediante un proceso de cálculos matemáticos sobre entidades geométricas tridimensionales producidas en un ordenador, y cuyo propósito es conseguir una proyección visual en dos dimensiones para ser mostrada en una pantalla o impresa en papel.

En general, el arte de los gráficos tridimensionales es similar a la escultura o la fotografía, mientras que el arte de los gráficos 2D es análogo a la pintura.

En los programas de gráficos por computadora esta distinción es a veces difusa: algunas aplicaciones 2D utilizan técnicas 3D para alcanzar ciertos efectos como iluminación, mientras que algunas aplicaciones 3D primarias hacen uso de técnicas 2D.

- **Pipeline grafico**

El funcionamiento de los sistemas gráficos standard es típicamente descrito como una abstracción llamada graphics pipeline (que en español se traduce como una tubería grafica pero seguirá siendo usado por su referencia en inglés), El término “pipeline” es usado en casi todos los ámbitos informáticos o en todo lo referente a procesamiento y en el caso grafico porque es el proceso necesario para la transformación del modelo matemático a los pixeles en la pantalla incluye varios pasos que son desarrollados en secuencia, donde los resultados de un paso o estación son puestos como valores de entrada del siguiente paso.

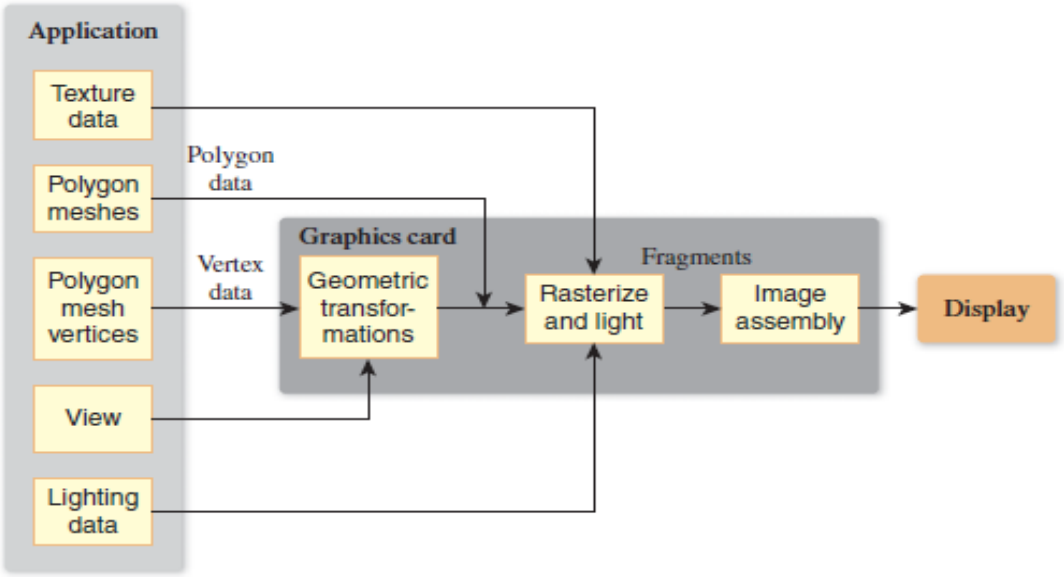


Ilustración 5: Imagen del pipeline grafico

Esta abstracción conocida como caja negra se utiliza en muchos aspectos del desarrollo de software ya que se busca que se puedan escribir programas ignorando las consideraciones físicas que varían de equipo a equipo siendo abierta dicha caja en procesos donde es valioso la eficiencia y el rendimiento.

Sistemas gráficos básicos:

En Un sistema grafico básico interactúan pocos dispositivos (un teclado y un mouse o una pantalla táctil, una CPU, una GPU y finalmente la pantalla, hoy en día estas pueden ser de diferentes tipo (CRT, LCD, OLED) los primeros consisten en un haz de electrones originado en la base de un tubo envasado al vacío es disparado hacia la pantalla, la cual tiene una capa hecha de un material de fósforo. Este fósforo se excita por el impacto de los electrones provocando un brillo de color rojo, verde o azul. La pantalla tiene miles de puntos llamados píxeles. Cada pixel es un impacto de electrones mezcla de rojo, verde o azul, y según la cantidad y fuerza del impacto brilla más o menos un color u otro pudiendo producir cantidad de colores. La mayoría de monitores CRT antiguos muestran algo de curvatura en las esquinas.

Los LCD constan de un panel plano de vidrio que es cubierto luego por una capa que contiene una rejilla con pequeños transistores. Estos transistores están agrupados en grupos de 3, y cada trio representa un píxel de la pantalla. La idea básica es que cuando se excitan con electricidad, estos transistores se pueden abrir

y apagar. Poniendo una luz detrás de la rejilla de transistores se pueden obtener imágenes.

Los OLED cuentan con pequeños diodos emisores de luz realizados con materiales semiconductores orgánicos que se encargan de crear las diferentes tonalidades de los colores que se desean mostrar.

Un programa típico se suele ejecutar sobre CPU, procesando las entradas de las interfaces de usuario y posteriormente es enviada la información a la GPU describiendo que debe ser mostrado en la pantalla

Algoritmos de Computación Grafica

- Algoritmo de Bresenham
- Algoritmo de Canny
- Algoritmo de relleno por difusión
- Algoritmo de Xiaolin Wu
- Algoritmo del pintor
- Algoritmo del punto medio para circunferencias
- Algoritmo del punto medio para elipses
- Algoritmos de Renderización
- Composición alfa
- Geometría computacional
- Grafo de vecindad relativa
- Método del conjunto de nivel
- Sombreado Gouraud
- Sombreado plano

2.3.2. Renderizado [9]

Desde el desarrollo y creación de las primeras computadoras, todo el proceso relacionado siempre ha tenido que ver con la técnica para interpretar los datos, con el pasar del tiempo fue siendo importante la representación de estos, para lo cual se crearon escenarios virtuales, estos tenían o tienen la característica que consumía mucho poder de computo, por tal motivo se han desarrollado técnicas que poco a poco fueron mejorando la apariencia de la imagen, en un principio se hacía una interpolación como se ve en los cursos básicos de ingeniería permitiendo tener un suavizado en las curvas, posteriormente se empezó a realizar mapeos de texturas y después se agregó brillos a los objetos que interactúan en la imagen hasta que finalmente llegan nuevas técnicas que con base a un modelo matemático que simula el comportamiento de los rayos de luz se hace posible obtener imágenes cada vez más reales y llamativas. Lo que ocurre con estas técnicas es que consumen una gran cantidad de poder de cómputo, por lo que la comunidad de síntesis de imágenes por computadora está en constante búsqueda de nuevas técnicas que permitan obtener imágenes foto realistas a un menor costo computacional, por consiguiente desde sus inicios siempre ha existido un duelo y se tiende a pensar que este va a continuar existiendo, la batalla consiste en mejorar los tiempos de ejecución pero cada día son mayores las resoluciones y tamaños de imágenes y/o videos que se utilizan.

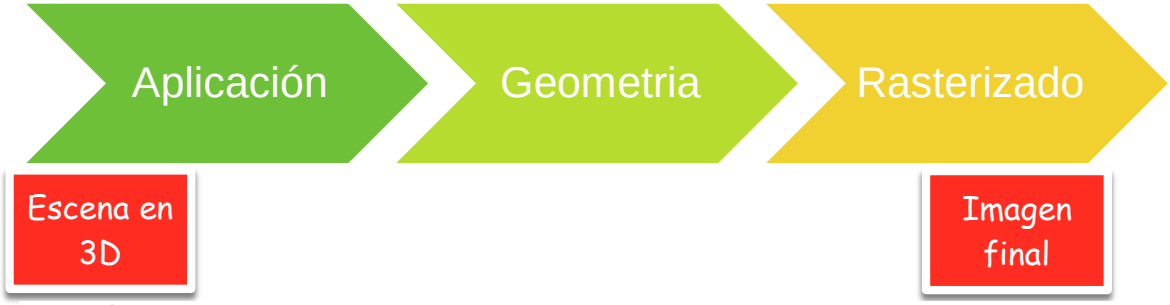


Ilustración 6: Secuencia del renderizado de una imagen

[9] Algoritmos óptimos para la generación de imágenes foto realistas, M. en C Mauricio Olguín Carbajal

Muchas de las propuestas para resolver la generación de graficas por computador, van desde modificar las estructuras de datos donde se almacena la información, pasando por editar la arquitectura empleada en busca de un mejor desempeño de los algoritmos hasta finalmente llegar a la forma de distribución de los rayos de luz, algunas de las propuestas antes mencionadas usan técnicas de transformaciones geométricas avanzadas que requiere de altos conocimientos matemáticos para el planteamiento de esa solución.

Básicamente se definen tres grandes etapas en un algoritmo de renderizado: Aplicación, geometría y rasterizado.

En la etapa de aplicaciones se realiza optimización usando:

- Nivel de detalle
- Objetos pre calculados
- Detección de colisiones
- Secuencias pre calculadas

Los algoritmos para el manejo del nivel de detalle, así como los objetos pre calculados la detección de colisiones y la secuencias recalculadas son algoritmos que tienen la finalidad de evitar círculos innecesarios de objetos o escenarios en donde se pueda encontrar redundancias y tiempo de procesamiento perdido que no va a tener relevancia en la imagen final, por ejemplo si se tiene objetos que no cambian de posición, orientación y/o tamaño, dichos objetos pueden ser pre calculados de manera que no impacten el tiempo de generación de una imagen, en donde existen otros objetos que si tienen variantes.

En el caso del nivel de detalle se hace que los objetos a grandes distancias tengan pocos detalles (lo que significa pocos polígonos) y a distancias cortas muy buen detalle aumentando así el desempeño del algoritmo de cálculo de la imagen evitando así el cálculo de cientos de polígonos que de objetos que están lejanos

Para la etapa geométrica se realiza optimización:

- División de espacios
- Arboles binarios
- Arboles octales
- Subdivisión adaptativa
- Algebra geométrica
- Algebra lineal
- Intersecciones

En la parte de procesamiento geométrico es posible apreciar que a nivel mundial se hace un gran esfuerzo para reducir el número de polígonos como las estructuras

necesarias para la descripción de los objetos y la relación que existe entre ellos y el escenario donde todos se encuentran. Típicamente los polígonos se describen como vectores y su manejo suele ser dado por matrices, pero no es este el único camino, existen otras formas donde se logra una reducción por medio del uso de metodologías diferentes a las comunes usadas para el manejo de los descriptores básicos de los objetos y el mundo donde interactúan.

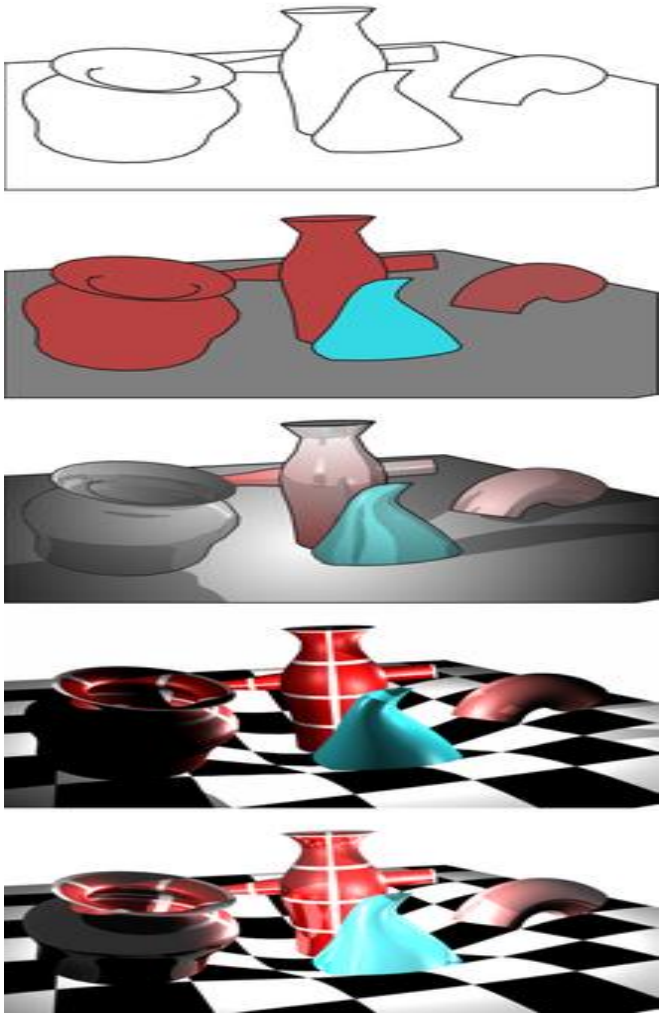
En esta etapa es donde se concentra gran parte del esfuerzo de optimización y se han usado inclusive soluciones heurísticas como parte de nuevas aproximaciones para resolver problemas como la programación geométrica, un desarrollo que se ha utilizado para las rotaciones en reemplazo de las matrices son los Cuaterniones (Extensión de los números reales similar a los números complejos), con lo cual se resuelto la rotación en objetos y escenarios tridimensionales de manera más eficiente

En la etapa de rasterizado se realiza optimización usando:

- Jerarquías Simples
- Meta jerarquías
- Distribución guiada de rayos
- Traza de rayos paralelos
- Clasificación de rayos
- Conos de rayos

A nivel de rasterizado también es posible usar una aproximación diferente para lograr combinar las dos grandes áreas de la resolución de visión y de iluminación, la traza de rayos y la radiosidad. Este concepto ya fue manejado por James T. Kajiya en su propuesta de la ecuación de renderización, en la que propone que ambos métodos mencionados anteriormente son, de hecho, una descripción diferente de la misma ecuación y propone la unión de ambos métodos a través de su ecuación ya sea por métodos convencionales o por medio de heurísticas, es aquí donde pueden entrar los algoritmos evolutivos para lograr encontrar la mejor solución práctica a la ecuación de renderizado. Y es que en la renderización ya se han usado heurísticas para solucionar algunos problemas específicos, por ejemplo para la distribución de rayos se usan mutaciones logrando una mejora notable en la imagen en el mismo tiempo [90] y [69]. Pero un algoritmo de generación de imágenes tiene una gran cantidad de parámetros a considerar y asimismo se han propuesto soluciones a cada uno de ellos en particular, tales como: la difracción de la luz en medios turbios [10], el transporte superficial de la luz para obtener una adecuada representación de subsuperficies, un manejo adecuado de la sombra y la penumbra, una adecuada iluminación global basada en la emisión de luz como una radiación ,

y la representación de texturas, entre muchos otros. Todos estos parámetros se van a solucionar de manera diferente dependiendo de la aproximación para la representación de la luz en el entorno de la computadora, las dos grandes tendencias para dicha representación son que la luz se comporte como una partícula representada por un rayo, es decir, una reflexión perfectamente especular (traza de rayos) y que la luz se comporte como una onda una reflexión perfectamente difusa (radiosidad), pero como menciona Kajiya, ambas soluciones son parte de una misma ecuación ya que en la realidad se tiene tanto una reflexión especular así como una reflexión difusa y es factible de realizar una solución general a la ecuación de renderizado, algunas propuestas son realizar de forma tradicional la solución combinando ambas técnicas como en el caso de la radiosidad con dos trayectorias y en el caso de la traza de rayos con un muestreo estocástico o con una generación de rayos hijos. Pero la solución de Kajiya propone el uso de un algoritmo muy parecido a la traza de rayos que genera reflexión difusa, reflexión especular y refracción, por medio del uso de un rayo que genera hijos de forma estocástica.



- **Rasterizado**

Ilustración 7: Imagen de un mismo modelo sobre diferentes técnicas de renderizado

El rasterizado[10] es también conocido como un mapa de bits o bitmap. Un gráfico o imagen que ha sido rasterizado es un fichero de datos que representa una matriz de píxeles (puntos de colores) denominada raster. En esta matriz el color de cada píxel es definido de forma individual. Se diferencia de los gráficos vectoriales porque estos almacenan la información en fórmulas matemáticas.

[10] Modeling the Interaction of Light between Diffuse Surfaces - Cindy M. Goral, Kenneth E. Torrance,

Los gráficos rasterizados al ser ampliados comienzan a pixelizarse, o sea, se agrandan los elementos constituyentes del gráfico, y pierden calidad. En cambio los gráficos vectoriales pueden ampliarse sin límites. Los gráficos rasterizados son útiles para imágenes fotográficas, las cuales no pueden ser representadas por vectores. La calidad de las imágenes rasterizadas es determinada por la cantidad de píxeles que poseen (Es lo que se conoce como resolución) y la cantidad de información en cada píxel (generalmente llamada profundidad de color).

Es decir, una imagen que almacena 24 bits de información por cada píxel (que es el estándar para todas las pantallas desde 1995) puede representar de forma más suave las tonalidades que una imagen que almacena sólo 16 bits por píxel. De igual manera en cuanto a la resolución, una imagen de 640 x 480 píxeles (Conocida como la resolución VGA que cuenta con un total de 307.200 píxeles constituyentes) se verá más accidentada y pixelizada comparada con una de 1280 x 1024 (que posee 1.310.720 píxeles constituyentes).

Una característica fundamental es que los gráficos rasterizado necesitan pasar por un proceso de compresión ya que suelen requerir muchos datos para poder almacenar imágenes de alta calidad. En su mayoría, las técnicas de compresión que buscan un tamaño menor, sacrifican información de la imagen para lograr este objetivo. Esto hace que la imagen pierda calidad y que esta no pueda ser recuperada.

Un ejemplo es que en la fotografía profesional los fotógrafos tienen la posibilidad de capturar las imágenes en dos formatos diferentes, uno de ellos es el conocido como .jpg el cual ya ha sufrido el proceso de compresión anteriormente descrito y existe otro formato conocido como RAW el cual almacena toda la información captada por el sensor de la cámara, aunque no es un gráfico vectorial tiene una mayor calidad que facilita el post operado o edición de las imagines.

- **Raytracing**

La técnica de raytracing[11] fue una de los primeros algoritmos de iluminación global en ser desarrollados. Este algoritmo reconoce que aunque billones de fotones deberían emitirse sólo los primeros que llegan al ojo son con los que forman la imagen resultante.

[11] http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html
[11] https://developer.nvidia.com/object/nvision08-IRT.html
[11]http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/modelosIlumionacion/raytracing_introduccion.html

El algoritmo trabaja trazando rayos como lo indica su propio nombre, los rayos van hacia atrás, desde cada píxel de la pantalla dentro de la escena. De esta manera es más eficiente, dado que así sólo se calcula la información que necesitamos para construir la imagen.
Para crear una imagen de esta manera, se realiza el procedimiento siguiente: Para cada píxel de la pantalla

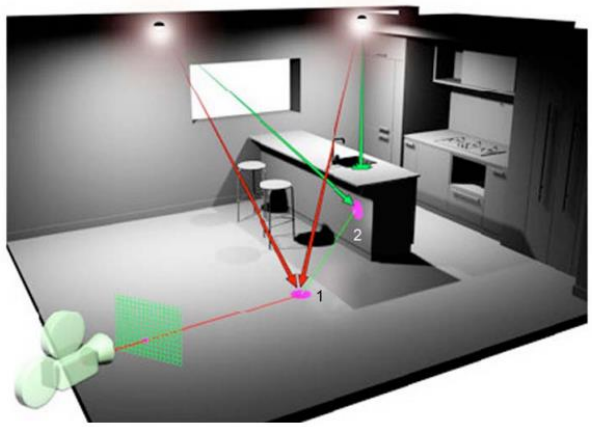


Ilustración 8: Imagen del comportamiento de la luz con Raytracing

• Primero. Se traza un rayo hacia atrás, desde la posición del ojo, a través del píxel del monitor, hasta que interseca con una superficie. La reflexión de la superficie es conocida gracias a la descripción del material, pero la cantidad de luz que refleja la superficie todavía nos es desconocida.

• Segundo. Se traza un rayo desde este punto de intersección en la superficie hacia cada origen de luz en la escena. Si el rayo que va a la luz no está bloqueado por ningún objeto, la contribución desde este origen se emplea para calcular el color de la superficie.

• Tercero. Si la superficie de intersección es brillante o transparente, también necesitamos determinar qué estamos viendo dentro o mediante la superficie que debemos procesar. Se repiten los procesos 1 y 2 en la dirección de reflejo (y en el caso de transparencia, se transmiten) hasta que se encuentra otra superficie. El color en los subsiguientes puntos de intersección se calcula y se junta en el punto original.

• Finalmente. Si la segunda superficie también es reflectiva o transparente, se repite el proceso de raytracing, y continuamos hasta un número máximo de iteraciones o hasta que no se encuentren más superficies intersecas.
Aunque la técnica de raytracing debería considerarse eficiente, ya que sólo se calcula la información requerida para generar la imagen, todavía es relativamente lenta para escenas que sean un poco complejas. Esta técnica es muy versátil y puede modelar un gran rango de efectos de iluminación.

- **Radiosidad**

A principios de los años sesenta, la investigación térmica desarrolló métodos para simular transferencia de calor radiado entre superficies. Aproximadamente veinte años más tarde, los investigadores en gráficos empezaron a emplear estas técnicas para modelar la propagación de la luz, hasta llegar a lo que hoy conocemos como radiosidad[12].

Esta técnica se diferencia de la técnica de raytracing en que fundamentalmente calcula la intensidad de todas las superficies del entorno, en lugar de calcular sólo las que han sido trazadas desde la pantalla. La idea en la que se basa la técnica de radiosidad es buscar el equilibrio de la energía que es emitida por los objetos emisores de luz y la energía que es absorbida por los objetos en el ambiente.

Para llevar a cabo este cálculo de iluminación, es necesario considerar que cuando la superficie de un objeto que no emite luz por sí mismo, es iluminada por otro objeto, ésta absorbe una cierta cantidad de la energía, pero refleja otra parte, por lo que puede ser considerada una emisora de luz por reflexión. De tal manera que todas las superficies en el ambiente son, de un modo u otro, emisoras de energía y, por lo tanto, cada una afecta a la iluminación de las demás superficies.

Cabe señalar que aunque la luz se refleja entre todos los objetos de la escena, el cálculo que realiza no es sobre las propiedades físicas reales, por lo que la representación que obtenemos normalmente no es físicamente real, sino una interpretación respecto a un algoritmo para generar la iluminación.

[12]http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/modelosIlumionacion/radiosidad_introduccion.html
[12] http://web.cs.wpi.edu/~matt/courses/cs563/talks/radiosity.html
[12] http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_1

Para calcular la iluminación con este algoritmo, se divide la superficie original en una malla de pequeñas superficies, referidas tales como elementos, así como en la Grafica #11.

El algoritmo de radiosidad calcula la cantidad de luz distribuida desde el foco de luz a cada elemento y desde cada uno de estos elementos a los diferentes elementos de la malla, donde el valor de radiosidad final generado se guarda como la suma de las aportaciones para cada elemento de la malla.

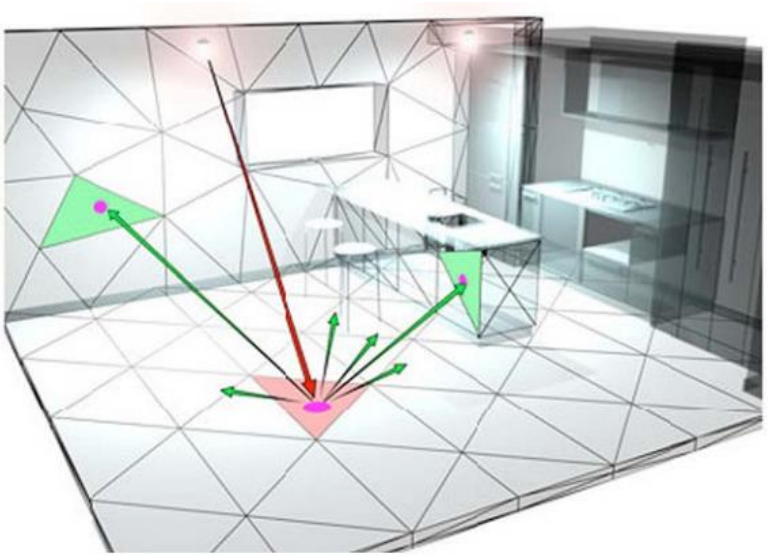


Ilustración 9: Imagen del comportamiento de la luz con Radiosidad

Esta técnica ha evolucionado mucho. En el año 1988 se inventó una técnica denominada refinamiento progresivo que mostraba los resultados de la radiosidad de manera visual e inmediata, y podíamos progresivamente ir refinando el cálculo hasta obtener una mayor calidad. Una década después inventaron una técnica denominada radiosidad por relajación estocástica (SRR), que construye una serie de posibles soluciones e intenta convergir hacia ellas (esta técnica es la base del algoritmo de radiosidad de 3dSMax).

Después de todos estos avances, es importante mencionar que, al igual que el raytracing, la radiosidad no produce todos los efectos de iluminación global y que también tiene sus propias aristas (por ejemplo, no se puede crear causticas). Sin embargo, entre los dos se complementan y trabajan bien. La radiosidad es excelente para renderizar reflexiones entre superficies difusas, mientras que la técnica de raytracing es excelente para calcular reflexiones especulares.

Una de las técnicas empleadas en el cálculo de la radiosidad es el método de Montecarlo para resolver este problema mediante números aleatorios y de forma estadística.

El auge de la radiosidad y otros métodos eficientes de renderización han posibilitado un auge en la infografía, siendo muy habitual encontrar por ejemplo películas que aprovechan estas técnicas para realizar efectos especiales.

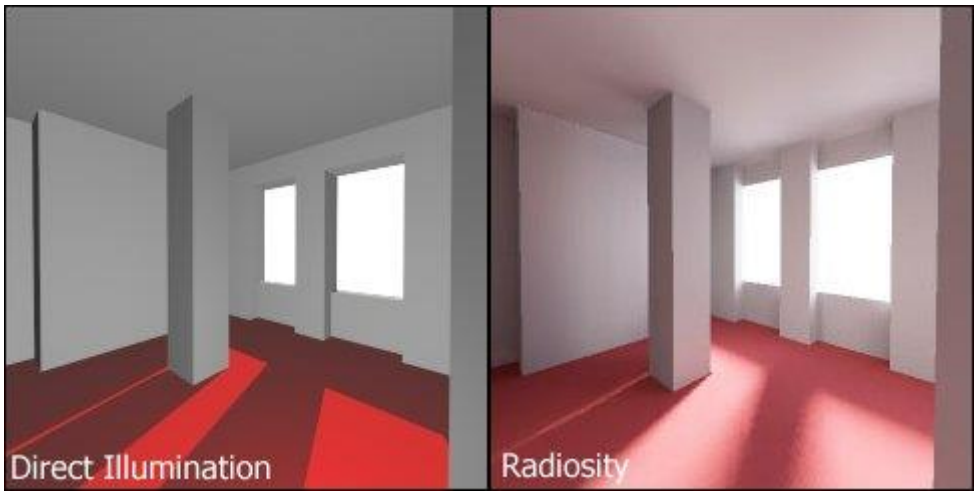


Ilustración 10: Imagen comparativa de la luz con la radiosidad y la luz directa

- **Seudocódigo de radiosidad:**

```
Cargar la escena
Dividir la escena en correctores de tamaño adecuado
iniciar_correctores:
        Para cada corrector en la escena si el corrector es una luz entonces
                corrector.emision = cantidad de luz
        Si no
                corrector.emision = cero
        corrector.excidente = corrector.emision
Finalizar bucle

Bucle_iteraciones:
Cada corrector recolecta luz de la escena
Para cada corrector en la escena
        Representar la escena desde el punto de vista de este corrector
corrector.incidente = suma de toda la luz incidente en la representación
Finalizar bucle_iteraciones

Calcular luz excedente de cada corrector:
Para cada corrector en la escena
        I = corrector.incidente
        R = corrector.reflectividad
        E = corrector.emision
        corrector.excidente = (I*R) + E
Finalizar bucle corrector
¿Hemos hecho suficientes pasadas?
Si no, volvemos a bucle_iteraciones
```

- **RayCasting:**

El RayCasting es un algoritmo que se utiliza en los gráficos por computador con la finalidad de simular de forma simplificada la interacción de la luz con los objetos. Una de sus características principales es que en vez de tomar como origen del proceso las fuentes de luz y tratar de emitir rayos en todas las direcciones, este lo que hace es tomar como punto de inicio al observador y desde ahí traza los rayos hacia la escena en busca de las fuentes de luz.

Como entradas el algoritmo utiliza la posición y las características físicas de cada objeto que forma parte de la escena, lo relacionado con la luz (ubicación, intensidad y tipo), la ubicación del observador y las dimensiones del plano

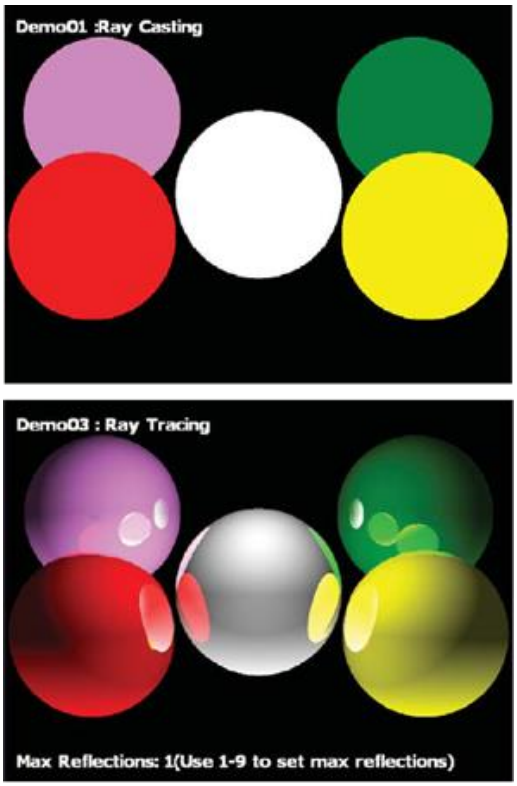


Ilustración 11: Raytracing vs Raycasting

- **Seudocódigo del RayCasting**

```
Para cada pixel de la imagen
 Crear un rayo desde el punto de visión a través del pixelActual
 Inicializar NearestT al INFINITO y NearestObject a NULL
 Para cada objeto de la escena
   Si el rayo se interseca con el objetoActual
     Si t de la intersección es menor que NearestT
        Poner NearestT = t de la intersección
        Poner NearestObject a objetoActual
 Si NearestObject = NULL {
    Rellenamos pixelActual con el color de fondo
 Sino
    Lanzar un rayo a cada foco de luz para comprobar las sombras
    Si la superficie es reflectora, generar un rayo reflectado (recursivo)
    Si la superficie es transparente, generar un rayo refractante (recursivo)
    Usar NearestObject y NearestT para computar la función de sombreado
    Rellenar este pixel con el color resultante de la función de sombreado
```

3. ANÁLISIS COMPRATIVO Y TOMA DE RESULTADOS

3.1. Toma de Muestras y diseño de pruebas

El desarrollo del proyecto se divide en dos fases; la primera de este consiste en el desarrollo del algoritmo de renderizado usando la técnica del RayTracing sobre CPU, es importante recalcar que aunque los procesadores actuales permiten paralelismo, para el caso práctico del proyecto solamente se ha ejecutado sobre un hilo de procesamiento, es decir, es un programa cíclico como la mayoría del software, con la finalidad de comparar las diferencias entre una ejecución cíclica y una paralela, la cual es la segunda fase del proyecto, el algoritmo de Raytracing pero ejecutado sobre GPU.

Las pruebas se han realizado en dos equipos con características diferentes pero ambos cuentan con procesadores de la marca Intel y las GPUs de la marca Nvidia utilizando la arquitectura que estas brindan para su desarrollo.

En cada equipo se han hecho pruebas con tres imágenes, cada una con diferentes características (mayor o menor cantidad de objetos) y se han realizado pruebas con 6 resoluciones por cada imagen (1024x1024 – 720x720 – 640x640 – 480x480 – 360x360 – 280x280) teniendo 18 tiempos por equipo y 36 tiempos en total para así poder tener una amplia muestra que facilite el análisis y el desarrollo de las conclusiones correspondientes necesarias para el proyecto.

3.2. Desarrollo Del Proyecto

El proyecto ha sido desarrollado utilizando las tecnologías de computación heterogénea, la cual tiene el objetivo de incrementar la capacidad de procesamiento de un dispositivo, ya sea un dispositivo móvil o un computador de escritorio, para el caso del proyecto la idea fue desarrollada sobre equipos de mesa, o equipos domésticos los cuales con el tiempo y la demanda multimedia han adquirido características elevadas que en muchos de los casos no utilizan completamente; por ejemplo podemos ver que muchos procesadores no se utilizan a su 100% ya que en las unidades de almacenamiento que generalmente son discos duros mecánicos presentan demoras o latencias en el proceso lecto-escritura necesario para el manejo de los datos, mientras que con el uso de unidades de estado sólido el cual funciona eléctricamente, esas velocidades de acceso a memoria se incrementa en más 10 veces a lo mecánico, con el caso de las GPUs encontramos

una cantidad de micro-procesadores que no se usan completamente salvo algunas aplicaciones como lo son los video juegos y aplicaciones de diseño multimedia (Photoshop, After Effects, Corel Draw, entre otros).

Para este proyecto se ha utilizado una librería de código abierto llamada *FreeImage* la cual soporta los formatos básicos de las imágenes y con la ventaja que permite el multithreading siendo a su vez compatible con arquitecturas tanto de 32 como de 64 bits y funciona con el sistema operativo Linux-Ubuntu 14-04 que ha sido el elegido para el proyecto, inicialmente se contaba con una distribución de elementary OS basado en el kernel de debían el cual en un principio funcionaba perfecto, pero con llegada de CUDA 6, la cual era compatible con las tarjetas gráficas que contábamos el sistema operativo ya no contaba con soporte para el compilador de esta nueva versión, que en el momento solo funcionaba con Ubuntu 14.04 por lo tanto se procedió con el cambio del sistema operativo.

Con todo el kit de herramientas de CUDA 6 viene incluido el IDE (Ambiente de desarrollo integrado) Nsight de eclipse el cual permite editar, construir y revisar las aplicaciones que se realizan en CUDA-C, inclusive la versión incluida permite el desarrollo remoto desde otro dispositivo; Se eligió esta versión de CUDA ya que ésta simplifica la programación con una memoria unificada la cual después de reservar la memoria necesaria en el dispositivo se encarga automáticamente de copiar los datos entre el host y el Device, adicionalmente permite una adaptabilidad multi-GPU que para cálculos pesados da una mejora en el rendimiento.

Se han hecho pruebas en dos equipos diferentes, tanto en CPU como en GPU los resultados varían con respecto a los recursos de cada equipo pero siempre se obtiene grandes mejoras en el uso de GPU sobre CPU.

- **Composición del Código:**

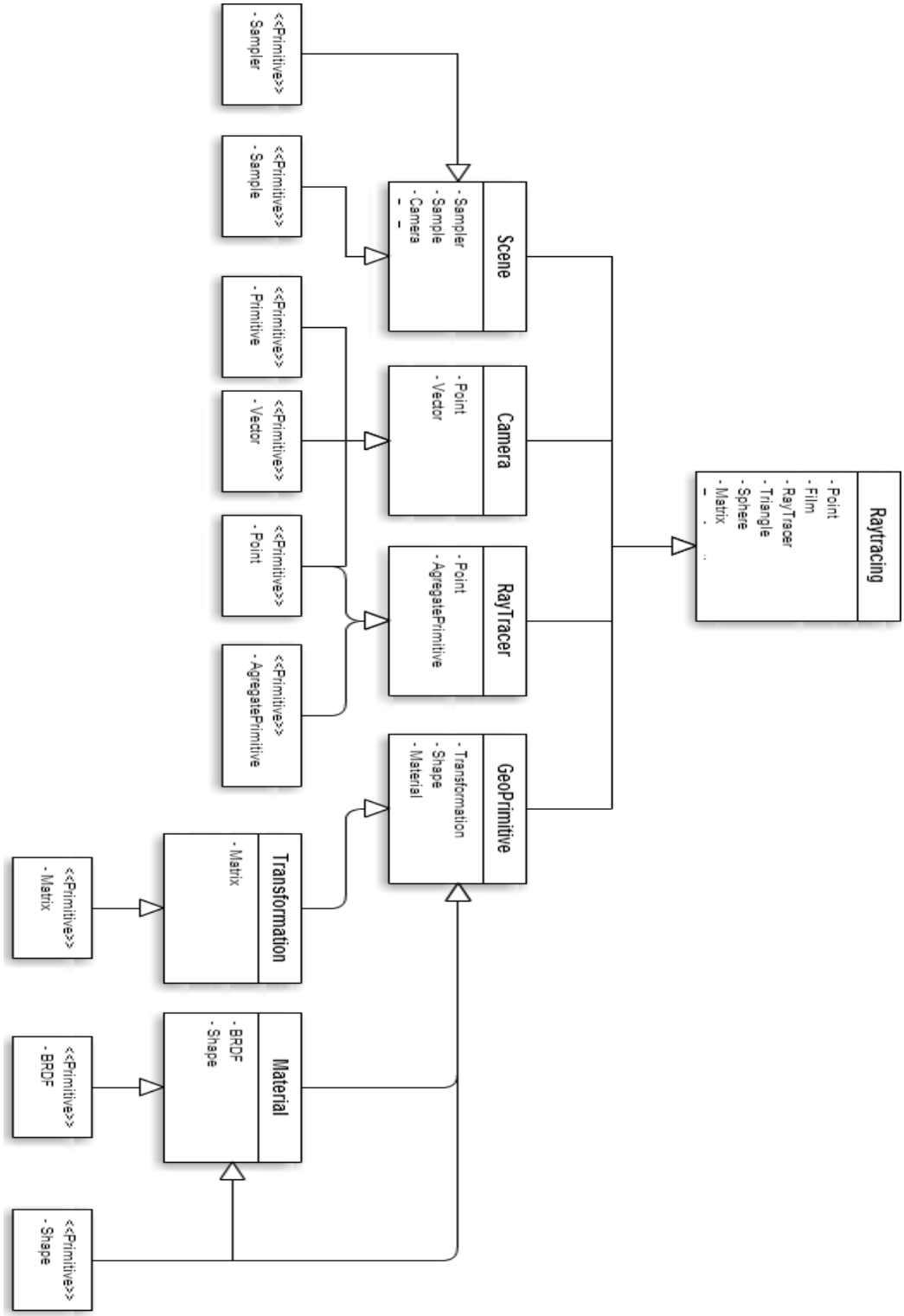


Ilustración 12: Composición del codigo

- **Código en CPU:**

Raytracing.cpp | Scene.cpp | RayTracer.cpp

```
void generateScene(float n)
{
    float startX = -6.5;
    float startY = 6.5;
    float startZ = -18;
    float u = 2.6;
    float sp = 50;
    float kr = 0;
    Color ka(0.1, 0.1, 0.1);
    Color ks(1, 1, 1);

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < n; k++)
            {
                float r = 1 - k / (n - 1);
                float g = j / (n - 1);
                float b = 1 - i / (n - 1);
                Color kd(r, g, b);
                Matrix move(startX + u * k, startY - u * j, startZ - u * i, 0, 0);
                Transformation trans(move);

                BRDF brdf(ka, kd, ks, sp, kr, 0);
                Material material(brdf);

                Sphere* ptrShape = (Sphere*) malloc(sizeof(Sphere));
                *ptrShape = sphere;

                Material* ptrMaterial = (Material*) malloc(sizeof(Material));
                *ptrMaterial = material;

                GeoPrimitive gpSphere(trans, trans.inverse(), &sphere, ptrMaterial);
                GeoPrimitive* ptrSphere = (GeoPrimitive*) malloc(sizeof(GeoPrimitive));
                *ptrSphere = gpSphere;
                auxGeo.push_back(gpSphere);
            }
        }
    }

}

int main(int argc, char* argv[])
```

Tab Size: 4 C++

Ilustración 13: Función en CPU para generar la Escena.

```
void Scene::render()
{
    while (sampler.generateSample(sample))
    {
        LocalGeo local;
        float thit;
        Ray ray;
        camera->generateRay(sample, ray);
        Color color(0, 0, 0);
        raytracer->trace(ray, 0, color);
        film->commit(sample, color);
    }
    film->writeImage();
}
```

Ilustración 6: Función principal del RayTracing donde se hace todo el proceso para cada pixel de la imagen.

Raytracing.cpp | Scene.cpp | RayTracer.cpp

```
26  void RayTracer::trace(Ray &ray, int depth, Color &color)
27  {
28      Color bg(0, 0, 0);
29      if (depth > max_depth)
30      {
31          color = bg;
32          return;
33      }
34
35      float thit = 0;
36      Intersection in;
37      if (!primitive->intersect(ray, &thit, in))
38      {
39          color = bg;
40          return;
41      }
42      BRDF brdf;
43
44      in.getPrimitive()->getBRDF(in.getLocal(), brdf);
45
46      Ray lray;
47      Color lcolor;
48      LocalGeo inlocal = in.getLocal()
49
50      for (int i = 0; i < lights.size(); i++)
51      {
52          lights[i]->generateLightRay(inlocal, lray, lcolor);
53          if (!primitive->intersectP(lray))
54          {
55              color = color + shading(inlocal, brdf, lray, lcolor);
56          }
57          else
58          {
59              Color base(255, 255, 255);
60              Color ambient = brdf.getKa();
61              Color diffuse = brdf.getKd();
62              color = color + base * ambient * diffuse;
63          }
64      }
65  }
66
```

Ilustración 15: Función del trazado de rayos.

En la versión sobre GPU el código cambia de la versión sobre CPU donde la clase **Scene** desaparece por el mismo poder que tiene la computación heterogénea donde todos los pixeles en este caso son operados de manera simultanéame, mientras que en la versión de CPU era necesario tener un ciclo que tomara cada uno de los pixeles y se hiciera todo el proceso de generar el rayo, hacer el trazado y almacenar los resultados obtenidos para posteriormente guardar la imagen.

La aproximación utilizada para resolver el problema de la parelelización del algoritmo fue asignar a cada hilo de ejecución de la GPU un pixel determinado a

renderizar e identificar el pixel correspondiente mediante las variables incluidas en CUDA *threadIdx* y *blockIdx* que proporcionan información sobre qué hilo y qué bloque se encuentra el proceso actual en ejecución (Ilustración 20).

Código sobre GPU:

```
    AggregatePrimitive primitives(list);

    //-------------------------------------------------------

    Film* d_film;
    Camera* d_camera ;
    RayTracer* d_raytracer;
    VectorLight* d_Llist;
    AggregatePrimitive* d_primitives;

    CUDA_CHECK_RETURN(cudaMallocManaged((void**)&d_film,sizeof(Film)));
    CUDA_CHECK_RETURN(cudaMallocManaged((void**)&d_camera,sizeof(Camera)));
    CUDA_CHECK_RETURN(cudaMallocManaged(&d_raytracer,sizeof(RayTracer)));
    CUDA_CHECK_RETURN(cudaMallocManaged((void**)&d_list,sizeof(list)));
    CUDA_CHECK_RETURN(cudaMallocManaged((void**)&d_Llist,sizeof(Llist)));
    CUDA_CHECK_RETURN(cudaMallocManaged((void**)&d_primitives,sizeof(primitives)));

    *d_camera = camera;
    *d_raytracer = raytracer;
    *d_film = film;
    *d_list = list;
    *d_Llist =Llist;
    *d_primitives = primitives;

    d_raytracer->addLights(*d_Llist);
    d_raytracer->addPrimitives(d_primitives);

    clock_t tStart = clock();
    render<<<height, width>>>(d_camera, d_raytracer, d_film);
    cudaDeviceSynchronize();
    printf("Tiempo de renderizado: %.2fs\n\n", (double)(clock() - tStart)/CLOCKS_PER_SEC);
    bitmap = FreeImage_Allocate(width, height, BPP);
    writeImage(d_film->getColorMap(), filename);
    FreeImage_Unload(bitmap);
    cudaFree(d_camera);
    cudaFree(d_raytracer);
    cudaFree(d_film);
    cudaFree(d_list);
    cudaFree(d_Llist);
    FreeImage_DeInitialise();

    return 0;
}
```

Ilustración 16: Inicialización y llamado a función principal en la GPU.

```
Point center(0, 0, 0);
Sphere sphere(center, 0.6);

VectorPrimitive list;

FIBITMAP* bitmap;

__managed__ int width; // = 1024;
__managed__ int height; // = 1024;

__global__ void render(Camera* camera, RayTracer* raytracer, Film* film)
{
    Sample sample(threadIdx.x, blockIdx.x);
    Ray ray;
    camera->generateRay(sample, ray);

    Color color(0, 0, 0);
    raytracer->trace(ray, 0, color);
    film->commit(sample, color);
    __syncthreads();
}
```

Ilustración 7: Función principal del RayTracing sobre GPU lanza todos los rayos simultáneamente.

- **Algunos resultados obtenidos sobre CPU:**

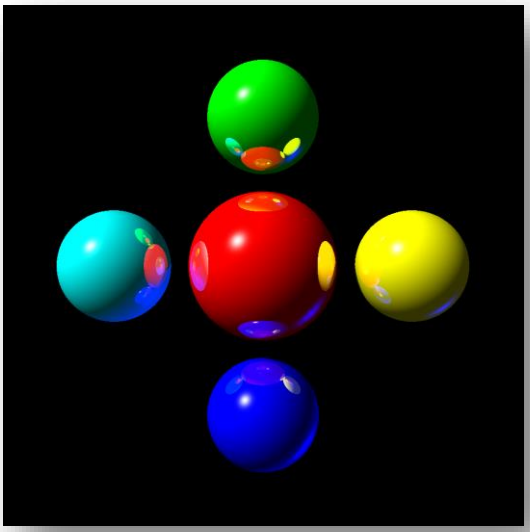


Ilustración 18: Imagen resultante probando la reflexión de la luz.

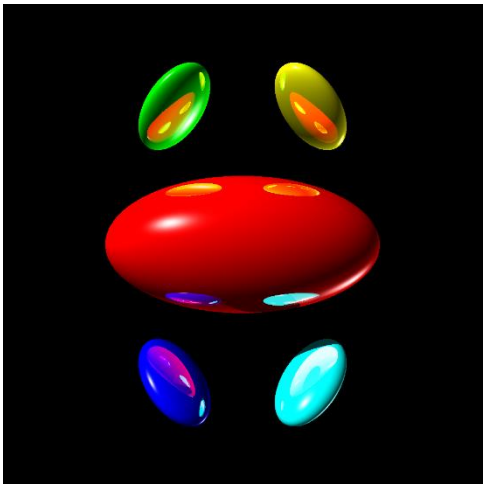


Ilustración 19: imagen probando la reflexión de la luz.

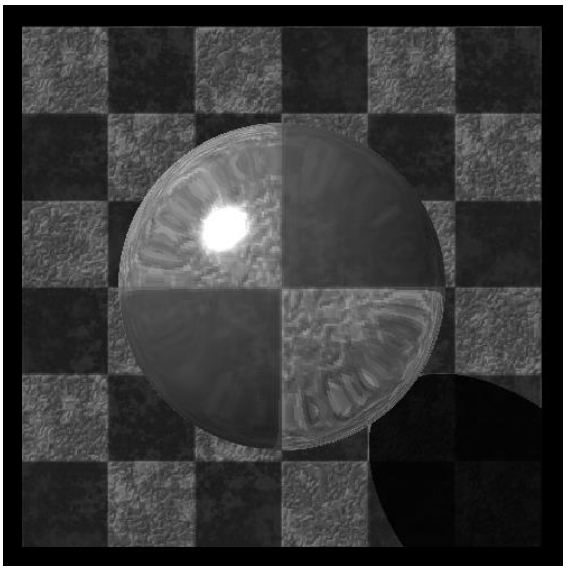


Ilustración 20: Imagen probando Texturas en los objetos.

Ilustración 21: Imagen resultante de probar diferentes texturas y reflexión en los objetos.

3.3. Resultados:

Para las pruebas del proyecto se han seleccionado 3 imágenes; las cuales han sido probadas en dos ambientes diferentes el primer ambiente se realiza en un computador con las siguientes características: Intel i5 4570, Board MSI Z87 G45, EVGA GTX 650, 4GB memoria RAM. Cada imagen es probada en diferentes resoluciones:

- ✓ 1024 x 1024
- ✓ 720 x 720
- ✓ 640 x 640
- ✓ 480 x 480

- ✓ 360 x 360
- ✓ 280 x 280

Donde los resultados para el ambiente 1 han sido los siguientes:

Imagen 1: test01.png

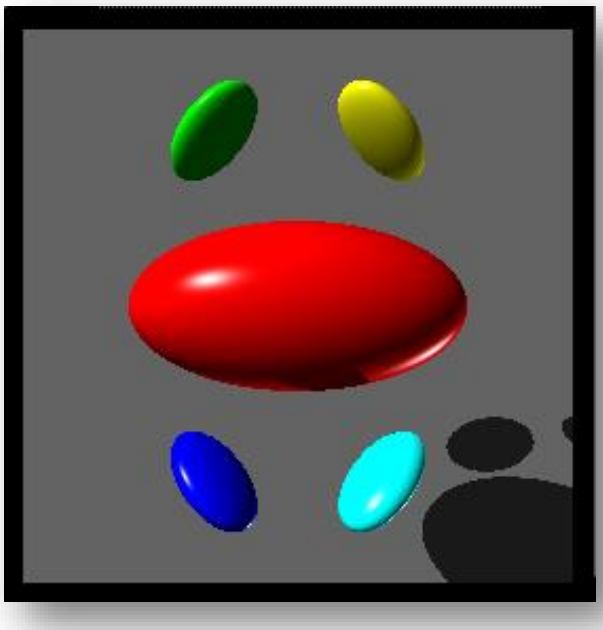


Ilustración 22: Imagen Test01.png

Test01.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	3,49	8,42	2,41
720 x 720	2,43	4,16	1,71
640 x 640	2,16	3,29	1,52
480 x 480	0,83	1,85	2,23
360 x 360	0,42	1,04	2,48
280 x 280	0,25	0,63	2,52
Promedio	1,59666667	3,231667	2,145466

Tabla 2: Resultados del algoritmo en Test01

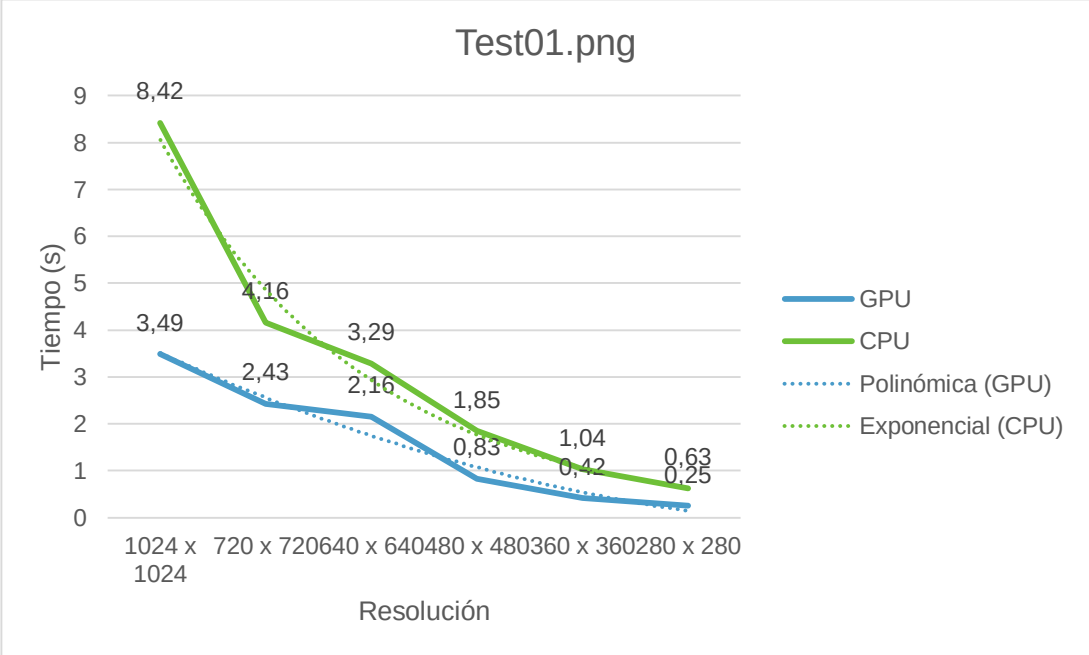


Ilustración 23: Grafico comparativo entre CPU y GPU de Test01

Imagen 2: test02.png

Ilustración 24: Test02

Test02.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	2,69	6,68	2,48
720 x 720	1,87	3,3	1,76
640 x 640	1,66	2,61	1,57
480 x 480	0,64	1,48	2,31
360 x 360	0,33	0,83	2,52
280 x 280	0,16	0,37	2,31
Promedio	1,225	2,545	2,16007

Tabla 3: Resultados del Algoritmo en Test02

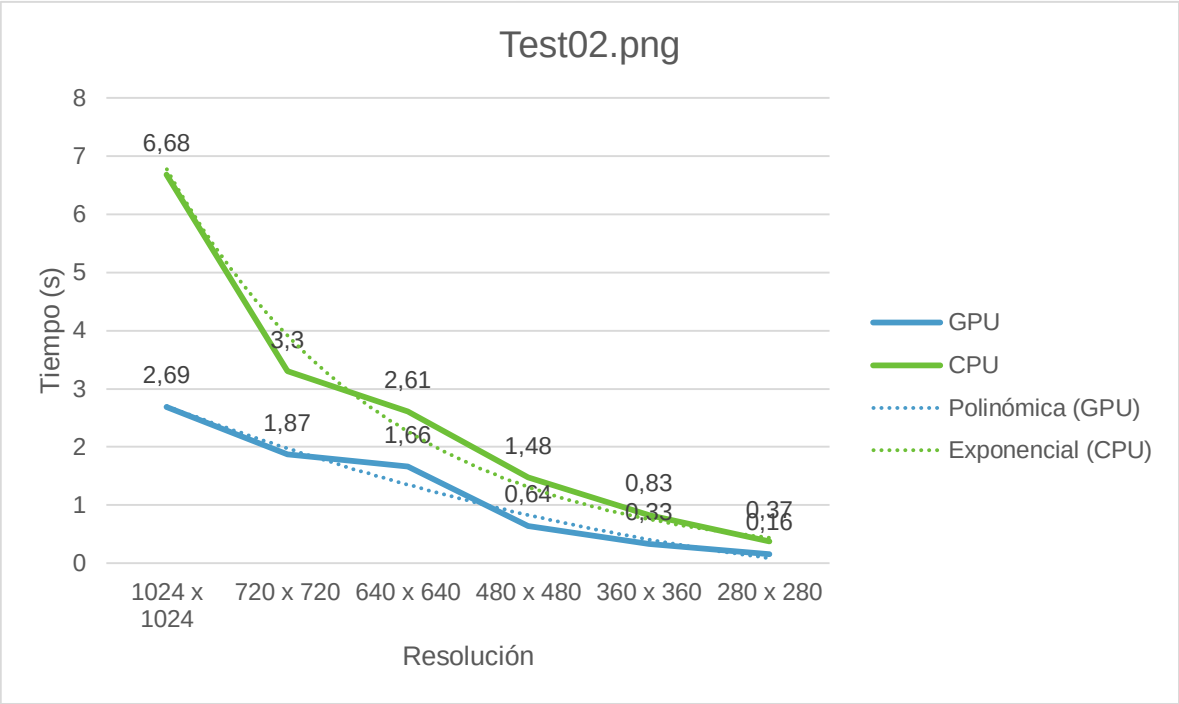


Ilustración 25: Comparativa de los resultados obtenidos entre CPU y GPU para la imagen Test02

Imagen 3: test03.png

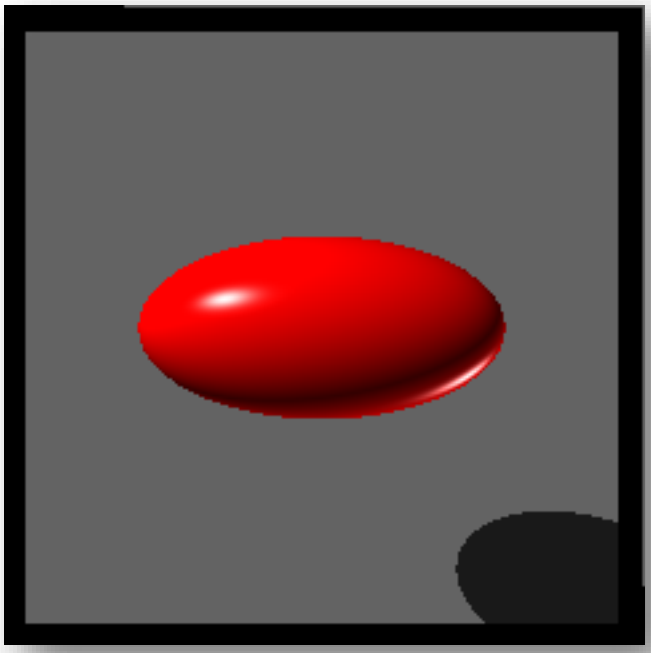


Ilustración 26: Imagen de Test03

Test03.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	1,91	4,87	2,55
720 x 720	1,32	2,41	1,83
640 x 640	1,18	1,9	1,61
480 x 480	0,45	1,07	2,38
360 x 360	0,23	0,61	2,65
280 x 280	0,11	0,27	2,45
Promedio	0,866667	1,855	2,24502707

Tabla 4: Resultados algoritmo en Test03

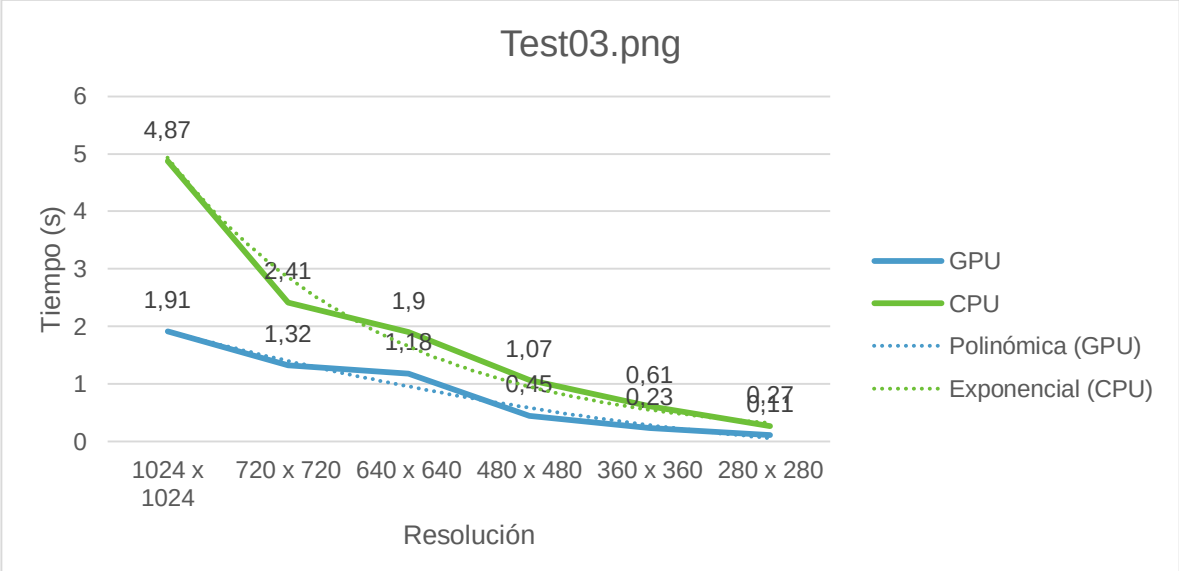


Ilustración 27: Comparativa de los resultados obtenidos entre CPU y GPU para la imagen Test03

Para el ambiente 2; el equipo cuenta con una Board Intel DX58SO, un procesador Intel i7 – 920, 6 Gb de memoria RAM y una GPU ASUS Geforce GTX 660 ti obteniendo los resultados siguientes:

Imagen 1: test01.png

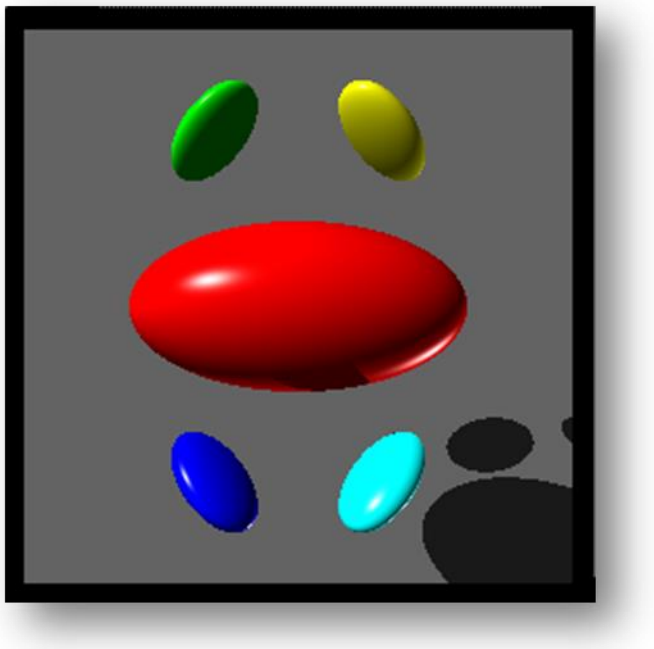


Ilustración 28: Imagen Test01

Test01.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	0,29	13,32	45,93
720 x 720	0,15	6,58	43,87
640 x 640	0,12	5,26	43,83
480 x 480	0,07	2,93	41,86
360 x 360	0,04	1,67	41,75
280 x 280	0,03	1,02	34,00
Promedio	0,11666667	5,13	41,87303

Tabla 5: Resultados del algoritmo en Test01 ambiente 2

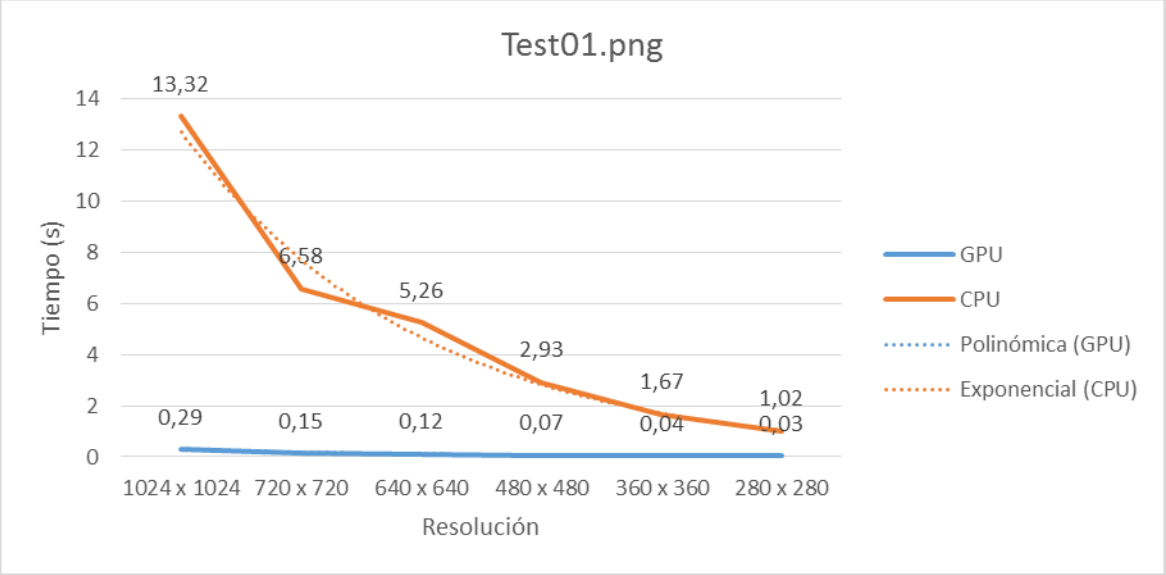
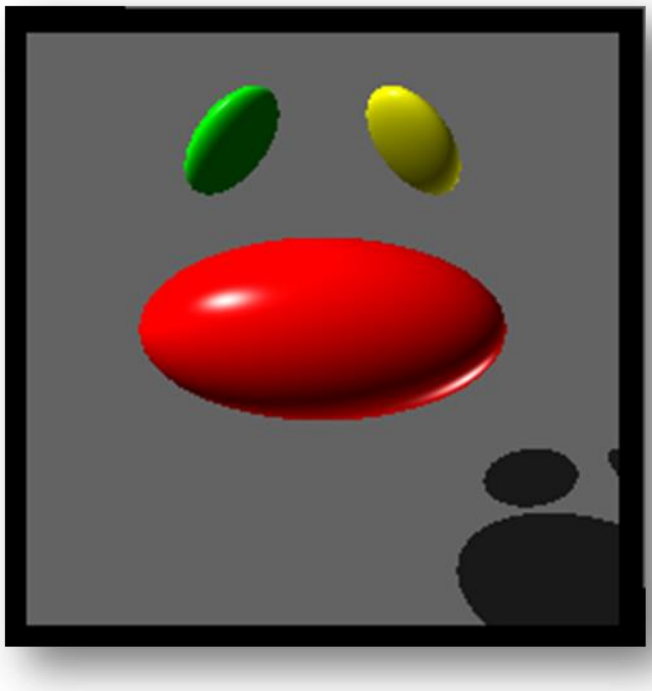


Ilustración 29: Comparativa test01 en ambiente 2

Imagen 02: test02.png

Ilustración 30:
Imagen Test 02



Test02.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	0,23	10,55	45,87
720 x 720	0,12	5,23	43,58
640 x 640	0,1	4,11	41,10
480 x 480	0,06	2,3	38,33
360 x 360	0,03	1,3	43,33
280 x 280	0,02	0,78	39,00
Promedio	0,093333	4,045	41,86993

Tabla 6: Resultados para Test 02 en ambiente 2

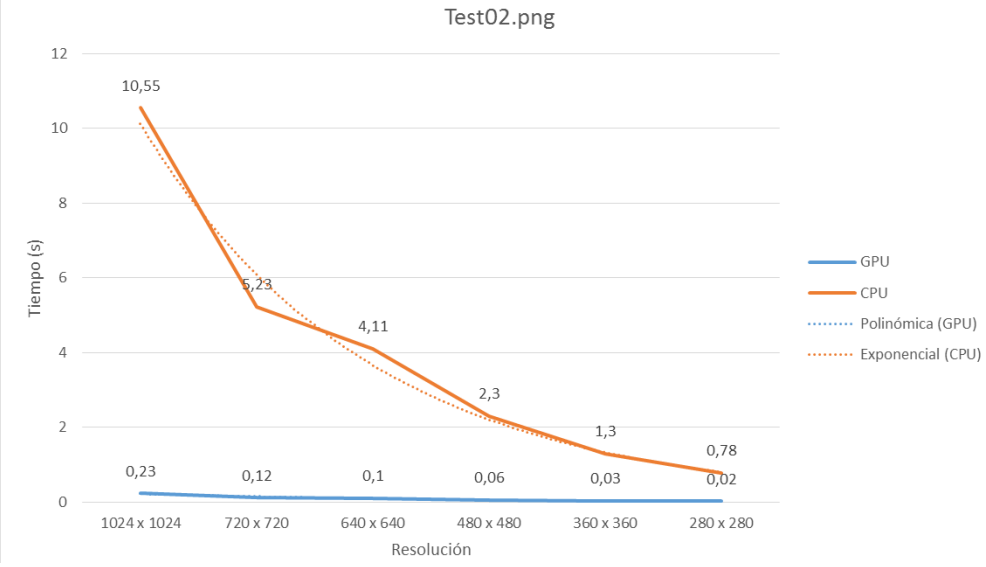


Ilustración 31: Comparación resultados Test02 en ambiente 2

Imagen 03: test03.png

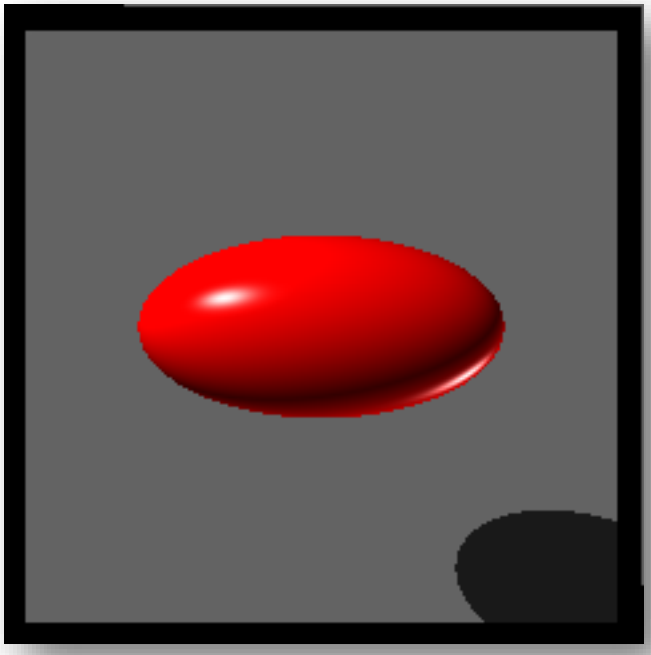


Ilustración 82: Imagen test03 ambiente 2

Test03.png			
Resolución	Tiempo (s)		Factor de Incremento
	GPU	CPU	
1024 x 1024	0,17	7,66	45,06
720 x 720	0,09	3,77	41,89
640 x 640	0,07	2,98	42,57
480 x 480	0,04	1,68	42,00
360 x 360	0,02	0,95	47,50
280 x 280	0,02	0,58	29,00
Promedio	0,068333	2,936667	41,3365235

Tabla 7: Resultados para test 03 en ambiente 2

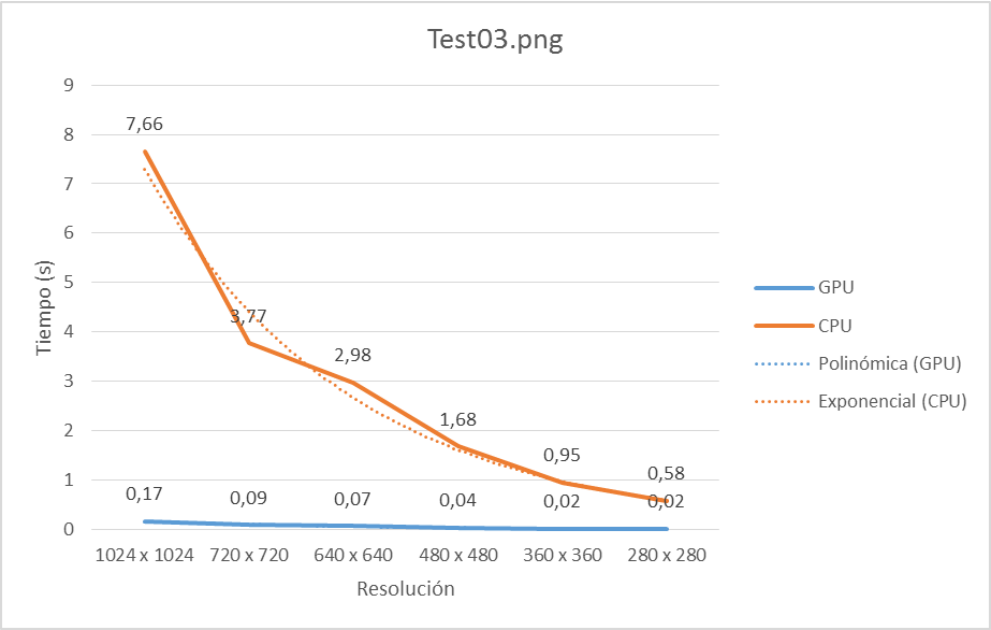


Ilustración 93: Comparación resultados teset02 ambiente 2:

Resultados Generales

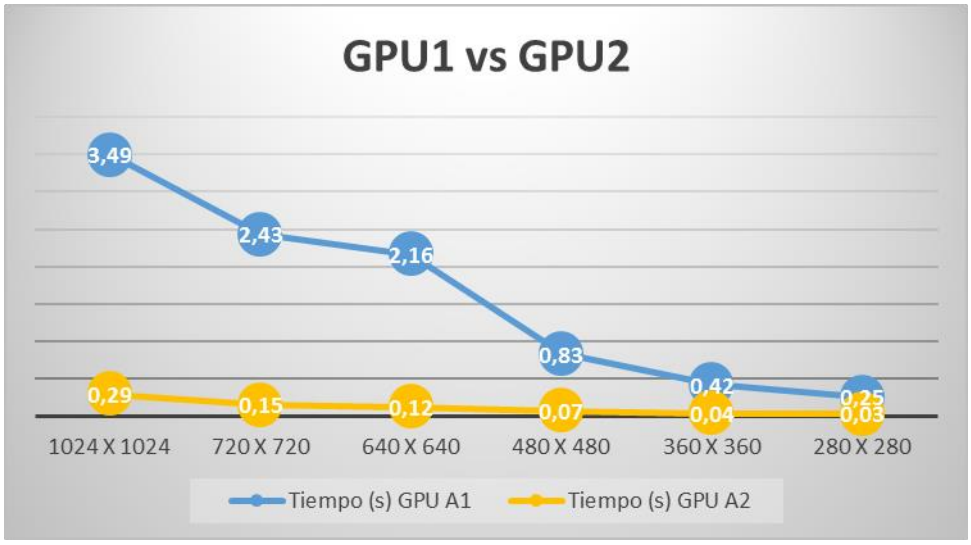


Ilustración 104: GPU1 vs GPU2

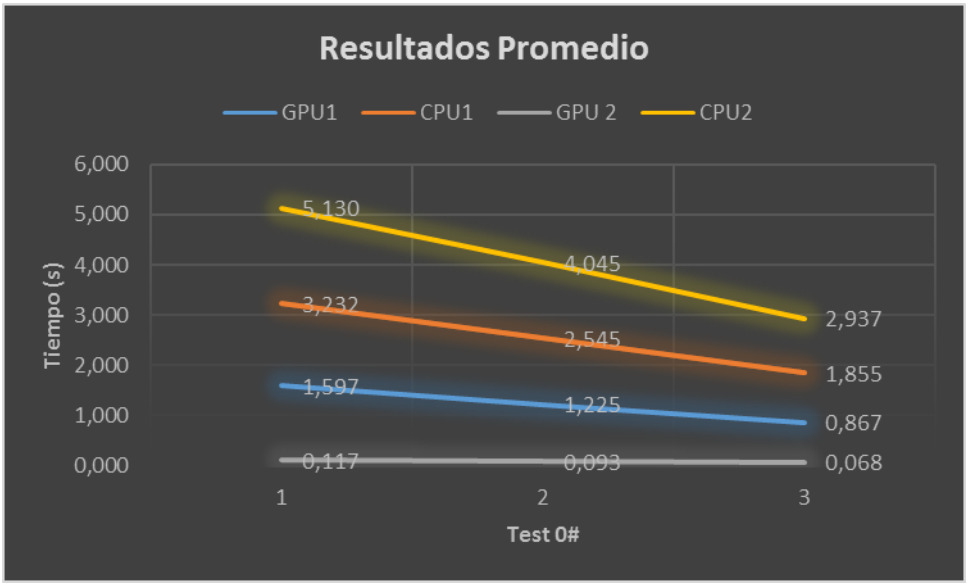


Ilustración 115: Resultados generales en ambos amientes

4. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

4.1. Conclusiones

1. El tiempo requerido para el procesamiento de la imagen sobre CPU va creciendo de manera exponencial entre mayor sea la resolución.

2. Se logró no solo cumplir lo planteado por la hipótesis inicial, en donde se propone que la ejecución de un algoritmo de raytracing consumía el doble de tiempo al ejecutarse sobre CPU que si este se ejecutara en GPU, donde en el análisis se encontró que en el mejor los casos la mejora superaba 47 veces los tiempos obtenidos; de igual manera, el tiempo de ejecución sobre CPU crece de manera exponencial a medida que se aumenta la complejidad de la imagen, mientras que corriendo dicho proceso sobre GPU el tiempo tiene una tendencia de crecimiento polinomial de grado dos.

3. Los tiempos de ejecución en la GPU se vieron disminuidos por lo menos en 1.52 veces lo requerido por la CPU en el ambiente 1 y 29.0 veces en el ambiente 2.

4. El uso de la GPU es recomendado para el procesamiento de grandes volúmenes de datos desde que las operaciones a realizar con esto sean de pocos accesos a memoria.

5. CUDA 6.0 maneja una memoria unificada lo cual da una mayor abstracción facilitando la programación y la comunicación entre el Host y el Device.

6. Es importante garantizar que toda la información que vaya a ser requerida por la GPU, se encuentre en la memoria unificada.

7. La programación paralela no permite la recursividad haciendo complejo la implementación de algunos efectos en los objetos.

8. Es posible mejorar los tiempos obtenidos optimizando completamente los accesos a memoria desde el dispositivo, y de igual manera con la CPU paralelizando según sea posible.

9. El proceso de paralelización requiere repensar los algoritmos buscando la manera de utilizar el paralelismo.

10. La optimización requiere tener una granularidad lo más fina posible, es decir, que cada hilo maneje un ítem de dato, reduciendo de esta manera latencias de acceso a memoria.

11. Es posible que algunos algoritmos sea mejor utilizarlos en CPU que en GPU, ya que en los accesos a memoria se encuentra el cuello de botella.

12. En la paralelización se logró mejorar los tiempos de ejecución, pero es importante resaltar que esas medidas están íntimamente relacionadas con la tarjeta gráfica utilizada, ya que la capacidad de cómputo de estas es una pieza clave en la variación de los resultados.

13. Adaptar el algoritmo al modelo que utiliza CUDA, implicó introducir algunos cambios drásticos, modificando el algoritmo original, pero se buscó conservar la forma en que se abordó el problema.

14. En ambos casos los resultados tuvieron mejores tiempos en la ejecución hecha sobre GPU, pudiendo considerar estos dispositivos aptos y eficientes para procesos de renderizado.

4.2. Recomendaciones

El planteamiento actual para la implementación del algoritmo limita las resoluciones posibles para las imágenes a 1024 x 1024 pixeles como máximo debido a que el máximo número de hilos posibles a lanzar simultáneamente en la versión de CUDA actual es de 1024.

La mayor carga de trabajo con la que se ha ejecutado el algoritmo durante las pruebas se encuentra en la resolución de las imágenes, sin embargo, es posible implementar un método que se encargue de generar una cantidad determinada de

objetos si se desea observar el comportamiento de los resultados con base a éste parámetro.

4.3. Trabajo Futuro

Con el desarrollo del presente proyecto, se han encontrado una serie de trabajos que pueden ser tratados posteriormente con la finalidad de ampliar investigación realizada. A continuación se describen los dos trabajos futuros más relevantes.

- Implementar el algoritmo de renderizado sobre GPU con el manejo de texturas, ya que actualmente CUDA no permite el manejo de estas.
- Ampliar el algoritmo para que permita el renderizado de otros tipos de objetos y modelar otros efectos físicos.
- Mejorar la implementación del algoritmo sobre GPU de manera que permita una renderización de imágenes con resoluciones mayores a 1024 establecida por la limitación actual de CUDA.
- Implementar un mecanismo que permita cargar modelos tridimensionales creados en software de diseño 3D actuales en un formato estándar.

5. BIBLIOGRAFIA

- Libro:
  AKL, Selim G. Diseño y análisis de algoritmos paralelos. 1992.

- Libro:
  BERT, J.; VISVIKIS, D. A fast CPU/GPU ray projector for fully 3D listmode pet reconstruction. IEEE Nuclear Science Symposium and Medical Imaging Conference, pp. 4126 - 4130, 2011.

- Libro:
  BIRN, Jeremy. Digital Lighting and Rendering (3rd Edition). 2013.

- Libro:
  FARBER, Rob, CUDA APPLICATION DESIGN AND DEVELOPMENT. 2011.

- Artículo:
  FOLEY, J.; VAN DAME, A.; FEINER, S., HUGHES, J.; PHILLIPS, R. Addison. Computer Graphics: Principles and Practice. Massachusetts: Wesley Publishing Company, 1996. 656 p.

- Libro:
  GLASSNER, Andrew S. An Introduction to Ray Tracing. Morgan Kaufmann, 1989. 327 p.

- Artículo:
  LAIDA, D. M. Ray tracing analysis of microwave link fading. IEEE. Fecha de conferencia: 30 de septiembre – 3 de octubre, 1990.

- Libro:
  LE GRAND, S. ; NICKOLLS, J. ; ANDERSON, J. ; HARDWICK, J. ; MORTON, S. ; PHILLIPS, E. ; Yao Zhang ; VOLKOV, V. Parallel Computing Experiences with CUDA. 2008.

- Web site:
  NVIDIA: <http://www.nvidia.com/object/cuda_home_new.html>
- Artículo:
  OCHSENFAHRT, U. y SALOMON, R. CREMA. A Parallel Hardware Raytracing Machine. IEEE. Fecha de conferencia: 27 – 30 de mayo, 2007.

- Libro:
  PHARR, Matt; HUMPHREVS, Greg. Physically Based Rendering, Second Edition: From Theory to Implementation. 2010.

- Libro:
  SANDERS, Jason; KANDROT, Edward. CUDA by Example. 2010.

- Libro:
  SHIRLEY, Peter; MORLEY, R. Keith. Realistic Ray Tracing, Second Edition. A. K. Peters, Ltd., 2003. 225 p.

- Libro:
  SUFFERN, Kevin. Raytracing from the Ground Up. A K Peters, 2007. 762 p.

- Libro:
  YOUNG, Chris; WELLS, Drew. RAY TRACING II. ANAYA MULTIMEDIA, 1995.

- Artículo:
  Cyril Zeller, Cuda c/c++. Barcelona, España,

- Libro:
  David B. Kirk, Web-mei W. Hwu, Programming Massively Parallel Processors

- Libro:
  Rob Farber, CUDA application design and development ISBN 9780123884268

- Artículo :
  Humberto Loaiza C, M.Sc, Introducción a los sistemas de visión en computadora

- Libro:
  John F. Hugues, Andries Van Dam, Morgan Mcguire, David Sklar, James Foley, Steven Feiner y Kurt Akeley, Computer Graphics, Third Edition, ISBN 9780321399526